Disentangled Syntax and Semantics for Stylized Text Generation

by

Yao Lu

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2020

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Neural network based methods are widely used in text generation. The end-to-end training of neural networks directly optimizes the text generation pipeline has been proved powerful in various tasks, including machine translation and summarization. However, the end-to-end neural network training makes it difficult to control the generation by partially changing the text properties (semantics, writing style, length, etc.). This makes text generation less flexible and controllable.

In this work, we study how to control the syntactic structure of text generation without changing the semantics. We proposed a variational autoencoder based model with disentangled the latent space. Our framework introduces various multitask learning and adversarial learning objectives as the regularization towards the syntax and content latent space, separately. The syntax latent space is required to parse a constituency tree while it cannot predict the bag-of-word feature of the given sentence. Likewise, the content latent space is required to predict the bag-of-word feature while contains no information for the parse tree.

Experiment results show that our model (TA-VAE) outperforms previous work. The quantitative and qualitative studies indicate that the TA-VAE model has a high-quality disentanglement of latent space for syntax controlled text generation.

Acknowledgements

First and foremost, I would like to thank my supervisor, Professor Olga Vechtomova, for her guidance and inspiring discussions during my study. She provides lots of feedback, which is indispensable for the thesis. I am grateful for her patience in teaching me how to carry out research. Besides, I would like to thank Dr. Lili Mou for stimulating discussions. I also want to thank Professor Pascal Poupart and Professor Jesse Hoey for providing feedback on my thesis.

I am lucky to meet so many friends at University of Waterloo: Ralph, Xinji, Yangtian, Qian, and Michael (knowing that I would miss someone). Special thanks go to Linqing. Thank you for all the accompany and support from college to graduate school.

Finally, I would like to thank my parents for their love and support throughout all these years.

Dedication

This is dedicated to my parents.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and motivation

Recently, data-driven end-to-end training of deep neural networks (DNNs) dominates the area of natural language processing (NLP) area. With massive training data, deep learning shows remarkable improvement towards various NLP application, e.g. machine translation [2, 12, 38], summarization [36] and question-answering systems [11, 14] etc. Though the neural network-based approach enables improvement of different areas compared to traditional approaches, there are still multiple issues.

First, the "data-hungry" properties of deep neural networks require lots of parallel training data. For example, the English-French machine translation [6] has millions of sentence pairs. In contrast, most NLP tasks do not have access to this due to massive human annotation efforts. The lack of access to sufficient data limited the development of lots of NLP areas.

Dataset	Task	Size
WMT14 English-French	machine translation	36 million
WMT16 English-German	machine translation	1.8 million
CNN-Dailymail [19]	Summarization	0.29 million
WikiSum [26]	Summarization	1.5 million
SQUAD [34]	Question answering	0.1 million

Second, the end-to-end training [12] performs direct optimization towards the training objective, which results in less flexible control towards the intermediate state. The only

feasible approach to change the generation style (e.g., sentiment, length) is to retrain the model on corresponding datasets. This leads to less flexibility for text generation as retrain a model takes lots of time and computation power. Therefore, how to enable models with controlled generation without retraining the model towards downstream tasks remains a crucial issue in NLP research.

Third, the neural-based models perform all transformation in vector space, make it difficult to explain the model's output. There is no way to point out each dimension in the latent space corresponds to what information. The lack of explainability further restricts the use of these models. Therefore, the disentanglement of the distributed representation is essential for the improvement of explainability.

To address the above issues, in this work, we focus on the exploration of controllable text generation without access to parallel data. More specifically, we seek an approach to manipulate the disentangled latent space of text representation to gain flexible control towards generative models.

1.2 Problem Definition

In this work, we study the problem of text style transfer. There is no strict definition of text style. Many factors (sentiment, formalism, and text length) may contribute to a specific style. To simplify the problem, we focus on syntax structure modeling, a well-defined property of natural language.

Given sentence pair $X = [w_1; w_2; ...; w_n]$ and $Y = [w_1; w_2; ...; w_m]$, where n and m denotes the length of the sentences X and Y, respectively. We project the sentence X into two different vector space, namely content space H_X^c and syntax space H_X^s. Likewise, for sentence Y, we also have H_Y^c and H_Y^s.

The objective of syntax transfer is to manipulate the H^s to generate sentences with different syntax structures. For example, we should be capable of generating a sentence $\hat{X}$ with the syntax template (H_Y^s) from sentence Y, while keeping the same content information (H_X^c) as sentence X.

Here are some examples of the syntax transfer.

As shown in Table 1.1, the objective of syntax style transfer is to maximize the overlap between generated sentence and content template while minimizing the distance between generation and syntax template in terms of syntax distance. The model needs to learn a good disentanglement between content and syntax information to perform well on this

Content Template	Syntax Template	Generation
a woman playing cards	two men are walking	a woman is playing cards
the child is in the garden	there is a dog	there is a child in the garden

Table 1.1: Examples of text syntax transfer

task. Besides, as mentioned in the Section 1.1, there is no existing dataset for the task. Therefore it is crucial to design a training strategy without access to annotated corpus.

To summarize, this task requires us to perform disentanglement between syntax and content information without explicit supervision signals (parallel annotated corpus).

1.3 Contributions

- We propose a model with explicit modeling of syntax information in latent space compared to vanilla VAE. Our approach shows remarkable improvement in precise control of the syntax structure for language generation.
- We use adversarial regularization in the latent space to encourage the disentanglement of syntax and content information. All the regularization relies on the properties of the text itself. Our model doesn't need to rely on a parallel corpus to learn the syntax transfer task.
- We propose a constituency tree regularization towards the latent space for better modeling of the syntax information. Unlike previous work of reconstructing the linearized tree, our designed layer-by-layer tree composition can encode explicit hierarchy information into the models, which is crucial for the syntax space manipulation. The use of full tree information is better than degenerated linearized tree representation according to the experiment result.
- We propose multiple metrics to quantitatively evaluate the performance of the syntax structure transfer task in terms of semantic distance, syntax distance, and generation diversity. We also create a standard evaluation set for the syntax transfer task on the SNLI dataset, enabling fair comparisons across different methods.

We also release the code and its corresponding model checkpoint and training hyperparameter configurations for reproducible studies.

Chapter 2

Background and Related Work

2.1 Language Model

The goal of language model is to estimate the probability of a sentence or a sequence of tokens $X = [x_1, x_2, x_3, ..., x_n]$. It can be used under the following scenarios:

- **Sentence scoring/ranking:** We can estimate the probability of a sentence $P(X) = P(x_1, x_2, x_3, ..., x_n)$. This probability can be used to measure the coherence or factual correctness of sentences.
- **Next token prediction:** we can estimate the probability of next token given an incomplete sequence by computing $P(x_{n+1}|x_1, x_2, x_3, ..., x_n)$. It is widely used in various language generation tasks (e.g., dialogue and summarization, etc.).

We use the chain rule to write computation of the joint probability of given the sequence X as $P(x_1, x_2, x_3, ..., x_n) = P(x_1)P(x_2|x_1)P(x_3|x_1, x_2)...P(x_n|x_1, x_2, ..., x_{n-1})$. To simplify, this can be written as

$$P(X) = \prod_i P(x_i|x_1, x_2, ..., x_{i-1}) \tag{2.1}$$

2.1.1 N-gram Language Model

A simple idea to estimate $P(x_n|x_1, x_2, ..., x_{n-1})$ is counting the occurrence of sequence $[x_1, x_2, x_3, ..., x_n]$ and $x_1, x_2, x_3, ..., x_{n-1}$, as shown in the following equation.

$$P(x_n|x_1, x_2, ..., x_{n-1}) = \frac{\text{Count}([x_1, x_2, x_3, ..., x_n])}{\text{Count}([x_1, x_2, x_3, ..., x_{n-1}])} \tag{2.2}$$

However, it is not feasible to get a good estimation towards that because the searching space is big, and there is no way to collect enough data to support this estimation. Therefore we need to introduce some assumptions to simplify this problem.

A widely used approach is to introduce Markov assumption into this conditional probability estimation. The basic idea of Markov assumption is that the current state is only relevant to a finite number of previous steps. This assumption can modify the Eq. 2.1 to the following, where k denote the finite step number relevant to the current state:

$$P(X) \approx \prod_i P(x_i|x_{i-k}, x_{i-k+1}, ..., x_{i-1}) \tag{2.3}$$

We rewrite the estimation of conditional probability in Eq. 2.2 as the following

$$P(x_n|x_1, x_2, ..., x_{n-1}) = \frac{\text{Count}([x_{n-k}, ..., x_n])}{\text{Count}([x_{n-k-1}, x_{n-k+1}, ..., x_{n-1}])} \tag{2.4}$$

The simplest form of Markov assumption is unigram language model, where $k = 1$ in equations mentioned above. We can also use bigram assumption to get better estimation.

We can extend k to trigrams, 4-grams, and even larger k. When the k is large enough, we can get the general form of the language model, as shown in Eq. 2.1. In that case, this probability estimation problem is almost impossible to solve in natural language processing. Therefore it is crucial to find the trade-off between the value of k and the computation cost.

The N-gram language model is not a perfect estimation of natural language as long-term dependency is crucial in language understanding. However, this approach is still widely used for the trade-off between accuracy and efficiency.

Here is an example of N-gram language model to illustrate how to estimate probability by counting the tokens. Given three sentences "[SOS] I am happy [EOS]", "[SOS] I do not like the food [EOS]" and "[SOS] So do I [EOS]". We use special token "[SOS]" and "[EOS]" to denote the start of sequence and end of sequence to ensure all probabilities sum up to one. We can compute the conditional probability by counting the occurrences. For example, we can get

$$P(\text{I}|[\text{start}]) = \frac{2}{3} = 0.667 \text{ and } P(\text{not}|\text{do}) = \frac{1}{2} \tag{2.5}$$

The evaluation of language model is crucial to select the proper N-gram for the language model computation. A comprehensive evaluation will plug the language model into

downstream tasks (e.g., speech recognition, machine translation, etc.), then compute the task-specific metrics (e.g., accuracy). However, this will be time-consuming and costly.

Therefore, an evaluation metric that does not involve external data is essential and convenient for evaluating language model quality. A widely used approach is perplexity, where we measure the language model performance on the language model test set instead of the downstream task test set.

The perplexity is the inverse probability over the test set, with proper normalization (e.g., number of tokens). It can be written as the following form

$$\text{Perplexity}(X) = \sqrt[n]{\frac{1}{P(x_1, x_2, ..., x_n)}} \tag{2.6}$$

or using replace the denominator with Eq. 2.1, we get

$$\text{Perplexity}(X) = \sqrt[n]{\frac{1}{\prod_i P(x_i|x_1, x_2, ..., x_{i-1})}} \tag{2.7}$$

For the counting-based estimation, the problem is the perplexity cannot handle the zero value, which means the above form will be infinity if encounter unseen N-grams. There are multiple smoothing techniques to overcome this issue. Here we introduce the most frequently used three approaches.

- Laplace smoothing: This approach adds a constant to each occurrence to avoid the divide by zero issue. In this case, the counting based bigram probability estimation is

$$P(x_n|x_{n-1}) = \frac{Count(x_{n-1}, x_n) + 1}{Count(x_{n-1}) + V} \tag{2.8}$$

  where V denotes the vocabulary size. This add-one strategy cannot work beyond bigram.

- Backoff: This strategy uses a smaller N-gram if the current N-gram statistics cannot provide sufficient information. For example, if no 4-gram is observed, we can use trigram, even bigram or unigram, until we have a reasonable estimation of this.

- Interpolation: This method performs fusion on different N-gram information with a weight factor. It can be written as

$$P(x_n|x_{n-1}, x_{n-2}) = \lambda_1 P(x_n|x_{n-1}, x_{n-2}) + \lambda_2 P(x_n|x_{n-1}) + \lambda_3 P(x_n), \quad st. \sum_i \lambda_i = 1 \tag{2.9}$$

We can use a validation set to get the optimal λ values by maximizing the probability of the validation set as follows

$$\log P(X|\lambda_1, \lambda_2, \lambda_3) = \sum_i \log P_{\lambda_1,\lambda_2,\lambda_3}(x_i|x_{i-1}) \tag{2.10}$$

2.1.2 Recurrent Neural Network Language Model

The N-gram language model is known for its lack of representation power, and it suffers from the data sparsity issue. Hence it is not an ideal model for high-quality text generation. Neural network is known for strong representation power [15]. It is an ideal choice for language modeling. In this section, we introduce Recurrent Neural Networks (RNNs), a powerful neural net family, to perform language modeling. For consistency, We will use the same notation as the N-gram language model section.

General Framework

The general framework of recurrent neural network language model [30] is shown in Figure 2.1. We will use the following notations in the computation of RNN language model.

- V denotes the vocabulary size.
- b denotes the embedding vector size
- $\mathbf{x}_n$ is the input token at time step n. We will convert it into one-hot vector, where the size of $\mathbf{x}_n$ is the vocabulary size.
- $\mathbf{e}_n$ is the corresponding distributed representation of token x_n
- $\mathbf{h}_n$ is the hidden state at time step n. We refer $n = 0$ as the initial hidden state.
- $\mathbf{E}$ is the embedding matrix. It will map the one-hot vectors into embedding space.
- $\mathbf{W}_e$ is the weight matrix perform transformation from embedding space to RNN hidden space.
- $\mathbf{W}_h$ is the transformation matrix between two consecutive hidden states.
- $\mathbf{b}_h$ is the bias term used in the transformation of hidden states.
- $\mathbf{U}$ performs the transformation from hidden state space to the output space.
- $\mathbf{b}_u$ is the bias term used in the transformation of output.

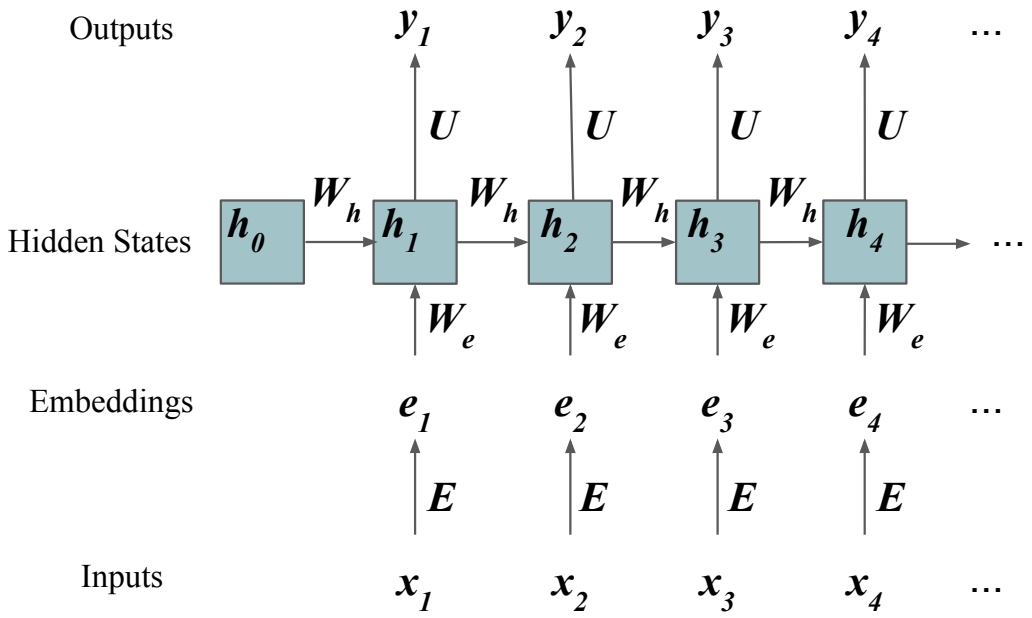


Figure 2.1: General framework illustration of recurrent neural networks

We will first generate a vocabulary based on the training set or use another preset vocabulary. Then convert the input sentence into a one-hot vector format.

The embedding vector of x_n can be achieved by embedding the dictionary lookup or multiplicating it with embedding matrix

$$e_t = x_t^T E \tag{2.11}$$

We then perform the transformation between previous time step $t-1$ hidden state and the embedding input e_t to get the current time step hidden state h_t using the following equation

$$\mathbf{h}_t = \sigma(\mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{W}_e \mathbf{e}_t + \mathbf{b}_h) \tag{2.12}$$

σ is the non-linear activation function, here we use the sigmoid function

$$\sigma(\mathbf{x}) = \frac{1}{1 + e^{-\mathbf{x}}} \tag{2.13}$$

For the first time step, we use the $\mathbf{h}_0$ as the previous step input, $\mathbf{h}_0$ is the initial state of recurrent neural modules, which can be random initialized.

Besides the hidden states, we also calculate the output by performing transformation of hidden states. In the language model next sequence prediction, we will not need each time step output, only the transformation of last time step hidden state is necessary to get the output. We can calculate the distribution over full vocabulary using the transformation as follows

$$\hat{\mathbf{y}}_t = \text{softmax}(\mathbf{U}\mathbf{h}_t + \mathbf{b}_u) \tag{2.14}$$

The softmax function normalizes all the terms to make them sum up to one. It can be interpreted as the probability.

$$\text{softmax}(\mathbf{x}) = \frac{e^{\mathbf{x}}}{\sum_i e^{x_i}} \tag{2.15}$$

As illustrated above, the advantages of recurrent neural network are:

- It can process variable-length input sequences.
- The model size does not increase with the sequence length, as all the parameters are reused for each time step.
- Unlike N-gram language model, the RNN architecture can encode all previous time steps information, enabling better language modeling capability.

Optimization

The training of RNN based language model is straight forward. We maximize the log-likelihood of the predictions at each time step. According to Eq. 2.14, we can get the distribution of vocabulary $\hat{\mathbf{y}}_t$ at time step t. We will compute the loss for this time step using cross-entropy

$$J_t(\theta) = \text{Cross-Entropy}(\mathbf{y}_t, \hat{\mathbf{y}}_t) = -\sum_{w \in V} \mathbf{y}_t^{(w)} \log \hat{\mathbf{y}}_t^{(w)} = -\log \hat{\mathbf{y}}_t^{(x_{t+1})} \tag{2.16}$$

This loss function measures the distance between the predicted distribution over vocabulary $\hat{\mathbf{y}}_t$ and the ground truth of next token x_{t+1}. The total loss can be written as

$$J(\theta) = \frac{1}{N} \sum_{t=1}^{N} J_t(\theta) = \frac{1}{N} \sum -\log \hat{\mathbf{y}}_t^{(x_{t+1})} \tag{2.17}$$

During the training, we try to minimize the negative log probability of the ground truth prediction at each time step. For each time step, the input can be the previous output or the ground truth. This two strategy can be summarized as

- Teacher Forcing: For each time step, we only use the ground truth as the input to get the prediction of next time step. This approach is widely used in RNN language model training.
- Scheduled Sampling [5]: This approach uses the model's prediction at time step t as the input to the time step $t+1$ computation. It reduces the exposure bias between training and inference, hence contributes to better performance.

In practice, we can use the mixture of teacher forcing and schedule sampling during the training. For each time step, we can use a preset probability to choose to use ground truth or model output as the next step input.

Since our next token prediction task only has the loss at the last time step, it involves backpropagation through time (BPTT) during optimization.

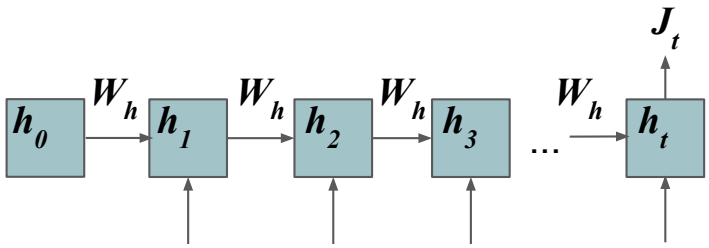


Figure 2.2: Backpropagation Through Time (BPTT)

According to multivariate chain rule, this can be derived as

$$\frac{\partial J_t}{\partial \mathbf{W}_h} = \sum_{i=1}^{t} \frac{\partial J_t}{\partial \mathbf{W}_h}\bigg|_i \tag{2.18}$$

Advanced Recurrent Nrural Network Module

The vanilla RNN framework introduced in Section 2.1.2 suffers from gradient vanish issue. In practice, we will use its advanced version by modifying the computation inside the recurrent modules. In this section, we introduce the Gated Recurrent Unit (GRU) [12].

Same as the setting of vanilla RNN, we have input $\mathbf{x}_t$ and $\mathbf{h}_t$ for each time step. GRU performs transformation as follows

$$\mathbf{u}_t = \sigma(\mathbf{W}_u \mathbf{h}_{t-1} + \mathbf{U}_u \mathbf{x}_t + \mathbf{b}_u) \tag{2.19}$$

$$\mathbf{r}_t = \sigma(\mathbf{W}_r \mathbf{h}_{t-1} + \mathbf{U}_r \mathbf{x}_t + \mathbf{b}_r) \tag{2.20}$$

$$\tilde{\mathbf{h}}_t = \tanh(\mathbf{W}_h(\mathbf{r}_t \circ \mathbf{h}_{t-1}) + \mathbf{U}_h \mathbf{x}_t + \mathbf{b}_h) \tag{2.21}$$

$$\mathbf{h}_t = (1 - \mathbf{u}_t) \circ \mathbf{h}_{t-1} + \mathbf{u}_t \circ \tilde{\mathbf{h}}_t \tag{2.22}$$

The update gate (Eq. 2.19) controls the hidden state updates or preservation. The reset gate(Eq. 2.20) controls the reuse of previous hidden state for computation. We then use the reset gate to select partial information from previous hidden states using Eq. 2.21. Finally, the hidden state for the current step can be computed using the update gate to integrate the information from both sides using Eq. 2.22.

Compared to other variants of RNNs (e.g., LSTM), GRU can achieve faster speed and fewer parameters. The design of GRU makes the gradient can be pass through longer sequences easier by introducing the reset gate. In practice, the gradient vanishing/exploding issue can be alleviated by using GRU. Therefore GRU can help model learn better long term dependency than vanilla RNNs. In this work, we will use GRU for all recurrent neural modules unless otherwise specified.

Advanced Recurrent Neural Network Layout

Besides the engineering of recurrent neural modules, there are lots of work in the layout design of RNNs for better representation power. We introduce two widely used layout designs in this section.

- Bidirectional RNNs: The vanilla architecture can only model a single direction (left to right). This may affect some tasks where the dependency appears in a different direction. A simple idea is to process the sequence from both directions, then concatenate the output together for downstream tasks. It does not apply to the next sequence prediction as we do not have access to the full sequence. Thus impossible to define the direction for incomplete sequences. This modification can be written as

$$\mathbf{h}_t^{\text{forward}} = \text{RNN}_{\text{forward}}(\mathbf{h}_{t-1}, \mathbf{x}_t) \tag{2.23}$$

$$\mathbf{h}_t^{\text{backward}} = \text{RNN}_{\text{backward}}(\mathbf{h}_{t+1}, \mathbf{x}_t) \tag{2.24}$$

$$\mathbf{h}_t = [\mathbf{h}_t^{\text{forward}}; \mathbf{h}_t^{\text{backward}}] \tag{2.25}$$

- Multi-layer RNNs: The deep RNN architecture may be capable of model complex, even hierarchical representation (e.g., lower layers for surface feature, higher layers for complex semantics). The stack multi-layer strategy has been widely used not only in RNNs but also in other neural architectures. We illustrate the diagram[1] in Figure 2.3. The input of the layer is the output of the previous layer.

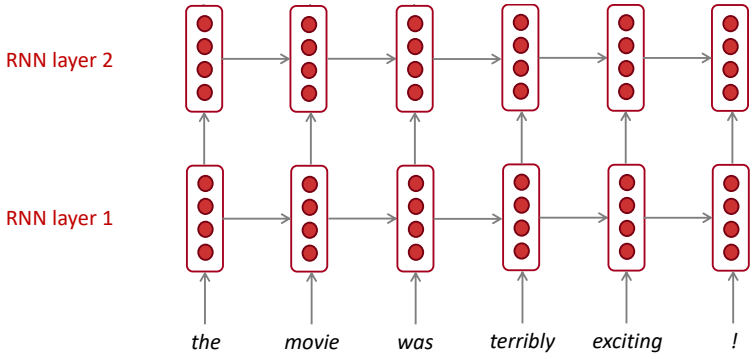


Figure 2.3: Multi-layer RNN Model

2.2 Sequence-to-Sequence Model

The language model mainly focuses on the next token prediction. It only covers a few parts of NLP applications. A wide range of NLP applications involves different types of input and output data. For example, the machine translation task requires converting sentences in the source language into the target language, where they have different vocabulary; the summarization task requires the summary of the long document input into short sentences, where the input and output length changes a lot. To address these tasks, we introduce the sequence-to-sequence model [2, 12] built on top of recurrent neural networks. This is the general framework for all the sequence tasks.

2.2.1 General Framework

The setting of sequence-to-sequence task requires input sequence $X = [x_1, x_2, x_3, ..., x_n]$ and target sequence $Y = [y_1, y_2, y_3, ..., y_t]$.

[1] This diagram is provided by Stanford CS224 course slide

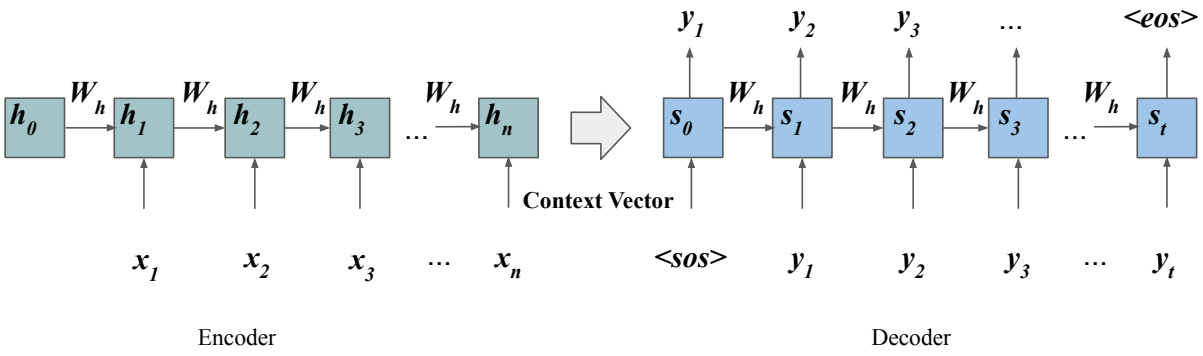


Figure 2.4: Sequence-to-sequence Framework

As shown in Figure 2.4, the sequence-to-sequence model involves two different parts: encoder and decoder. The encoder and decoder are two different recurrent neural networks, as described in Section 2.1.2.

The encoder takes source text X, then encodes it into a fixed-sized vector (context vector). This context vector can be used as the initial state for the decoder RNN. Decoder is an RNN language model for the generation of target sequence Y. It can be formulated as

$$P(Y|X) = P(y_1|x_1, x_2, ..., x_n)P(y_2|y_1, x_1, ..., x_n)...P(y_t|y_{t-1}, y_{t-2}, ..., y_1, x_1, ..., x_n) \quad (2.26)$$

The learning objective of sequence-to-sequence model is similar to Eq. 2.16 and Eq. 2.17. We refer readers to Section 2.1.2 for learning and optimization details.

2.2.2 Advanced Context Modeling

The vanilla sequence-to-sequence model uses the last hidden state of encoder as the initial hidden state of decoder. It may result in the issue that the last hidden state is not powerful enough to encode all necessary information for the decoding process. Decoding based on only single vector information may be problematic as there exists a gap between the encoder input and decoder target. It is helpful for modeling by introducing a connection between them other than only rely on the context vector.

A simple but effective method is to attend a specific part of the source sequence during the decoding process. This method is called attention [2]. The core idea of attention is to

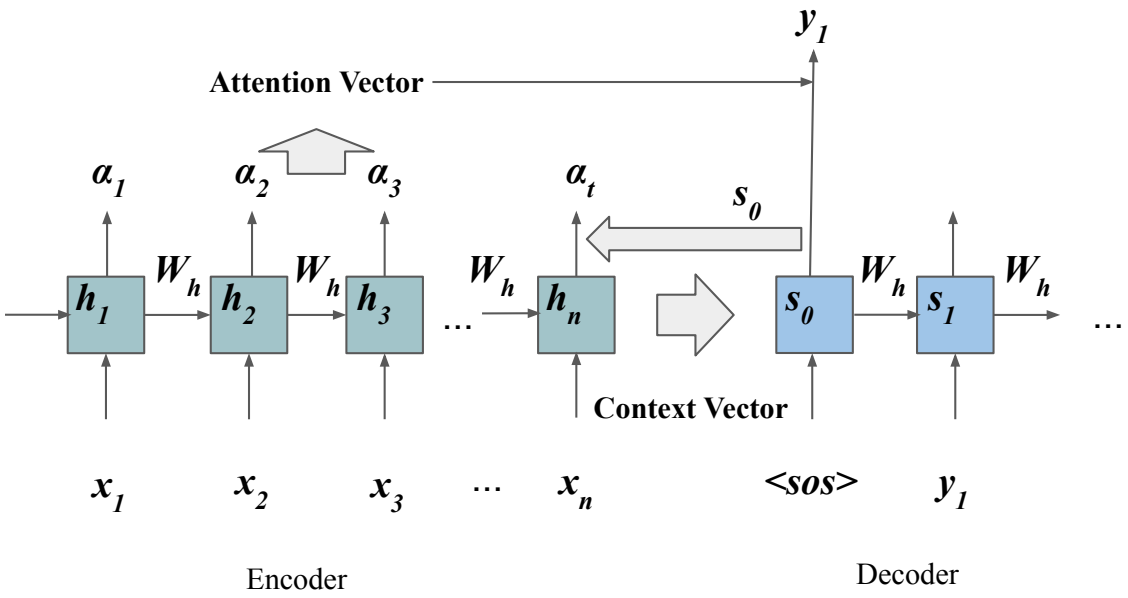


Figure 2.5: Attention in Sequence-to-sequence Model

attend to different parts of the input sequences dynamically. We can get soft alignment and better dependency modeling across input and target.

For each time step t in decoding, we have decoder state $\mathbf{s}_t$ and all the encoder states $\mathbf{h}_1, \mathbf{h}_2, ..., \mathbf{h}_n$. The attention score can be computed by multiply current decoder state with all encoder states.

$$\mathbf{e}_t = [\mathbf{s}_t^T \mathbf{h}_1, \mathbf{s}_t^T \mathbf{h}_2, ..., \mathbf{s}_t^T \mathbf{h}_n] \tag{2.27}$$

We then normalize the attention scores to the attention distribution

$$\alpha_t = softmax(\mathbf{e}_t) \tag{2.28}$$

Finally, we use the attention distribution as a weighted sum over all encoder states to compute the attention output for the decoding process

$$\mathbf{a}_t = \sum_{i=1}^{n} \alpha_i^{(t)} \mathbf{h}_i \tag{2.29}$$

We then concatenate the attention vector and the decoder output for the sequence prediction. It is worth mention that the attention vector keeps changing for each decoding

time step (Eq. 2.27). For each decoding time step, we can have a unique attention map over the input sequence. The advantages of introducing attention into sequence-to-sequence model are following

- Attention can provide better explainability with the soft-alignment with the input sequence. For the machine translation task, we can visualize the attention map to understand the parallel alignment between different language tokens.
- The attention alleviates the bottleneck by using only a vector to pass information between encoder and decoder. The decoder can direct use the information at each encoding step, therefore result in better sequence modeling performance.
- Attention-based sequence-to-sequence model can handle much longer sequences without suffering from gradient vanish issue. Thus enables the extremely long sequence modeling in this area.

2.2.3 Advanced Decoding Strategy

Some recent works show that recurrent neural network has chaotic behavior, and sometimes get trapped in local area [24]. It will result in the repeated generation issue. The greedy decoding strategy tries to maximize the output probability at each time step often suffers from this incoherent generation.

An ideal method is to perform an exhaustive search over all possibilities. We denote the vocabulary size as V, target sequence as T. This approach has $\mathcal{O}(V^T)$ complexity. It is impossible to do that in practice.

A trade-off approach only keeps track of a finite number k of partial generated sentences with the highest probability. We call the number k as beam size, where the k usually be a number less than 10. Those partial generated sentences are named hypotheses. We will select the best hypothesis based on the log probability

$$\text{score}(\text{hypothesis}) = \sum_{i}^{T} \log P(y_i | x_1, x_2, ..., x_n, y_1, y_2, ..., y_{i-1}) \tag{2.30}$$

In some decoding tasks, we can even add more heuristics for the hypothesis scoring. For example, we can set the score of sentences with repeated N-gram to zero. We can also add length penalty towards this score function to encourage length controlled generation.

2.3 Autoencoder

Autoencoder is an unsupervised/self-supervised training diagram. An autoencoder includes encoder f and decoder g. It takes the input $\mathbf{x}$ and maps it to an intermediate representation. Then reconstruct the input using the intermediate representation information. The principle of autoencoder can be written as

$$g(f(\mathbf{x})) = \mathbf{x} \tag{2.31}$$

The intermediate representation usually has a much smaller dimension size in comparison to the input. Therefore the intermediate can be forced to encode the essential information of the input.

For example, if we use the linear form for function f and g, where

$$\begin{aligned} f(\mathbf{x}) &= \mathbf{W}_f \mathbf{x} \\ g(\mathbf{x}) &= \mathbf{W}_g \mathbf{x} \end{aligned} \tag{2.32}$$

The reconstruction of input $\mathbf{x}$ can be written as $\hat{\mathbf{x}} = \mathbf{W}_g \mathbf{W}_f \mathbf{x}$. The learning objective for the reconstruction of $\mathbf{x}$ is

$$J_{\text{reconstruction}} = \sum_n \|\hat{\mathbf{x}}_n - \mathbf{x}_n\|_2^2 \tag{2.33}$$

This form is equivalent to the principal component analysis (PCA) if the objective function is mean square loss. Therefore we can view the autoencoder as a general form of information compression.

The linear function can also be replaced with non-linear function. For simplicity, we will use the notation $f(\mathbf{x}; \mathbf{W}_f)$ and $g(\mathbf{x}; \mathbf{W}_g)$. As long as there is a non-linear component in f and g, this autoencoder can be viewed as one layer neural network. We can also replace this neural architecture into recurrent neural network as described in Section 2.1.2.

For the better learning of the latent representation, we can impose more regularizations.

- sparse regularization: we can enforce the sparsity of the latent vector by introducing L_1 penalty towards the learning objective.

$$J_{\text{reconstruction}} = \sum_n \|\hat{\mathbf{x}}_n - \mathbf{x}_n\|_2^2 + \lambda \|f(\mathbf{x}_n; \mathbf{W}_f)\|_1 \tag{2.34}$$

- denoising autoencoder: To make the model robust to random noise, we can use the perturbed $\mathbf{x}^{'}$ instead of original $\mathbf{x}$ as the input. The learning objective is

$$J_{\text{reconstruction}} = \sum_n \|g(f(\mathbf{x}_n^{'}; \mathbf{W}_f)); \mathbf{W}_g) - \mathbf{x}_n\|_2^2 + \lambda \|f(\mathbf{x}_n; \mathbf{W}_f)\|_1 \tag{2.35}$$

2.3.1 Variational Autoencoder

All the above forms of autoencoder only emphasize the robustness, sparsity of the latent representation. In other words, they mainly focus on the feature extraction using autoencoder, and then fine-tune on downstream tasks, especially in discriminative tasks.

The idea of variational autoencoder [23] is to enforce the latent representation to be a known distribution (e.g. Gaussian distribution $\mathcal{N}(\mathbf{z}; \mathbf{0}, \mathbf{I})$). We can sample from the distribution to get a latent vector, then use it for generation task, as shown in Figure 2.6.

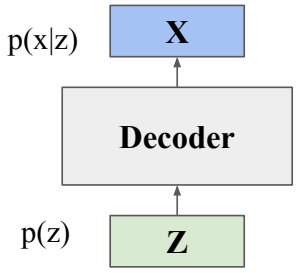


Figure 2.6: Generation from Prior

For simplicity, We will use ϕ and θ to denote the parameters of the VAE encoder and decoder instead of using one-layer fully-connected notation shown in previous section.

We can choose the prior $p(z)$ to be a simple distribution. In our work, we set it to Gaussian distribution. We may be able to use more complicated distributions. However, a multivariate Gaussian should be sufficient for most modeling tasks in most cases.

The optimization of this model is to maximize the likelihood of the training data,

$$p_\theta = \int p_\theta(\mathbf{z}) p_\theta(\mathbf{x}|\mathbf{z}) \mathrm{d}\mathbf{z} \tag{2.36}$$

However, this equation is intractable to compute for all possible $\mathbf{z}$. The posterior density $p_\theta(\mathbf{z}|\mathbf{x}) = p_\theta(\mathbf{x}|\mathbf{z}) p_\theta(\mathbf{z}) / p_\theta(\mathbf{x})$ is also intractable. The solution is to use encoder network $q_\phi(\mathbf{z}|\mathbf{x})$ to approximate $p_\theta(\mathbf{z}|\mathbf{x})$. Here is the diagram of VAE model[2].

We use neural network to approximate the mean and covariance instead of sampling directly. This is called reparameterization trick. It can make the whole framework differentiable. With the above modification, we can rewrite the log-likelihood of data as

$$\log p_\theta(\mathbf{x}_i) = \mathbf{E}_\mathbf{z}[\log p_\theta(\mathbf{x}_i|\mathbf{z})] - \mathrm{D}_{KL}(q_\phi(\mathbf{z}|\mathbf{x}_i) \| p_\theta(\mathbf{z})) + \mathrm{D}_{KL}(q_\phi(\mathbf{z}|\mathbf{x}_i) \| p_\theta(\mathbf{z}|\mathbf{x}_i)) \tag{2.37}$$

[2]This diagram is based on modification from cs231n Stanford course slides

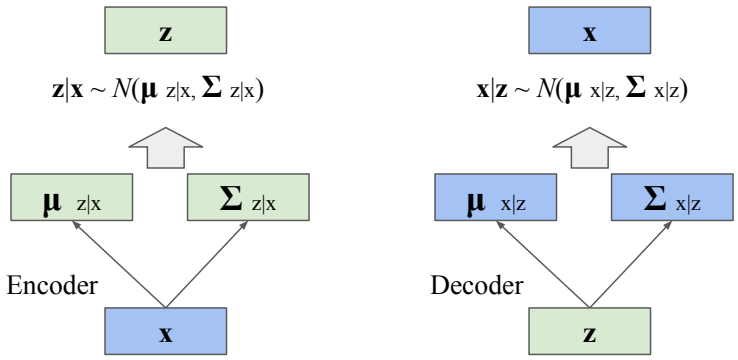


Figure 2.7: VAE framework

The first term can be estimated using sampling with the reparameterization. The second term is the KL divergence between the posterior and prior. The last term is intractable, while it can be ignored as it always has a positive value. If we set the first term and second term as our learning objective, this is tractable and differentiable everywhere. The first term is essentially reconstructing the input data, while the second term is to make the approximate posterior close enough to the prior. Therefore, our learning objective (evidence lower bound; ELBO) is

$$J_{VAE} = \sum_i \mathbf{E}_{\mathbf{z}}[\log p_\theta(\mathbf{x}_i|\mathbf{z})] - \mathrm{D}_{KL}(q_\phi(\mathbf{z}|\mathbf{x}_i)\|p_\theta(\mathbf{z})) \tag{2.38}$$

The optimization is similar to what we introduced in section 2.1.2, where we can use stochastic gradient descent to maximize the evidence lower bound.

2.4 Parsing and Tree based Models

Learning hierarchical tree structure is an essential topic in natural language processing. This structure can help to reduce the ambiguity of language. This is useful for various tasks, including natural language inference and coreference resolution.

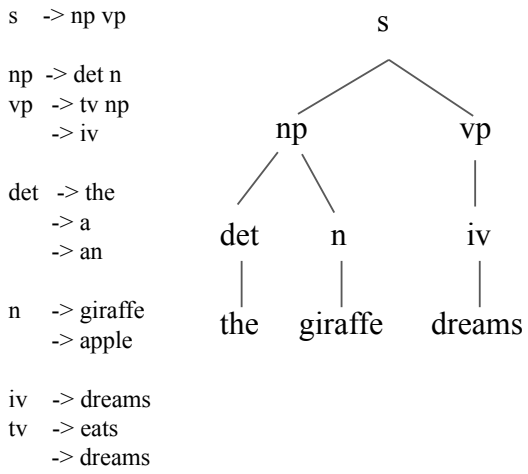
2.4.1 Parsing

Context Free Grammar

Context-free grammar (CFG) defines a set of rules and assigns each word a corresponding syntax tag. For each sentence, we can then recursively apply the rules then get a tree structure. CFG is not only useful in natural language processing but also in programming language.

For formal notation, we can write the CFG as tuple $< \mathbf{N}, \Sigma, \mathbf{S}, \mathbf{R} >$.

- **N** denotes the set of non-terminals as shown in the right Figure 2.8.
- Σ denotes the set of terminals (words).
- **S** denotes the start symbol (e.g. ROOT).
- **R** denotes the set of rules as shown in the left of Figure 2.8.



denotation	description
np	noun phrase
vp	verb phrase
s	sentence
det	determiner (article)
n	noun
tv	transitive verb (takes an object)
iv	intransitive verb
prep	preposition
pp	prepositional phrase
adj	adjective

Figure 2.8: Example of CFG

The pre-defined rules can guarantee we can have a reasonable parse tree for sentence. However, this kind of rules can only cover a few sentences. For many sentences, there will be no parse tree with incomplete rules. Therefore, we need to relax the constraint to make it possible to parse more sentences.

Probabilistic Context Free Grammar

The probabilistic context free grammar (PCFG) introduceS additional distribution q towards the vanilla form of CFG. For each $rin\mathbf{R}$, and $X \in \mathbf{N}$, we will have

$$\sum_{\alpha \to \beta \in R: \alpha = X} q(\alpha \to \beta) = 1 \tag{2.39}$$

For example, take the $vp- > tvnp$ and $vp- > iv$ in right part of Figure 2.8, the PCFG will assign probability p_1 and p_2 to these two rules, with constraint that $p_1 + p_2 = 1$.

The learning of PCFG can be performed by counting all labeled sentences to get the probability distribution. This is similar to the n-gram language model counting based estimation, as introduced in previous sections.

2.4.2 Tree-based Recursive Neural Models

The recurrent neural networks process sentence in the left-to-right and/or right-to-left manner. This may not be optimal for tasks sensitive to local syntax structure changes or sentences with extremely long dependencies. This issue can be alleviated using the tree structure, two components may be far from each other in the sequential order, but very close in the tree.

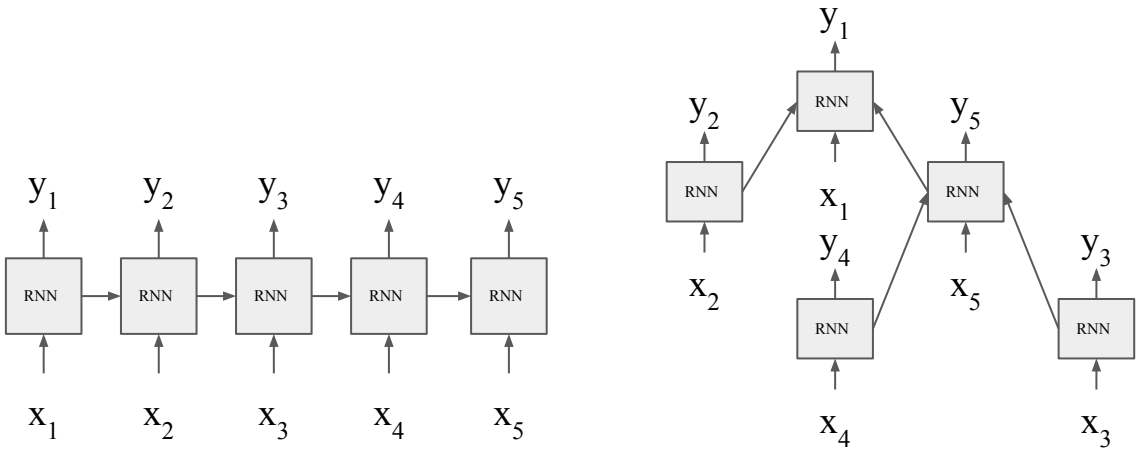


Figure 2.9: Comparison between RNN and Tree-RNN

Unlike previous method encode the sequence in the original order, TreeRNN [39] proposed a method using parse tree as guidance for the encoding process. As shown in right part of Figure 2.9, the model encode the sequence in to order of $x_3 \rightarrow x_4 \rightarrow x_5 \rightarrow x_2 \rightarrow x_1$ instead of original sequence (left part) $x_1 \rightarrow x_2 \rightarrow x_3 \rightarrow x_4 \rightarrow x_5$.

The architecture is almost the same as tradition RNN models, the main difference is that TreeRNN requires the sum of all children's hidden state as input for computation $\mathbf{h}_i^{\text{input}} = \sum_{j \in Child(i)} \mathbf{h}_j$, while the traditional RNN only requires $\mathbf{h}_i^{\text{input}} = \mathbf{h}_{i-1}$. Besides, we also need to use the children's information with proper forget gate transformation.

The advantage of this architecture is the explicit modeling of long term dependency. However, this modeling heavily relies on the quality of the parse tree. In some extreme cases, the parse tree may mislead the prediction if this task is not sensitive to local structures. Furthermore, all the recurrent/recursive based models suffer from inefficiency issues. These architectures can hardly perform parallel computation, resulting in slower speed compared to convolution neural networks.

2.5 Related Work

Variational autoencoder was first proposed by Kingma et al. [23]. They proved that the reparameterization of variational lower bound could be optimized using gradient-descent based approach. Thus makes it possible to model the data distribution with an approximate inference model. VAE has been widely used for image generation [35, 32, 40] and get comparable result to generative adversarial networks(GANs) [17] and its variants [1, 9].

Bowman et al. [7] extended the VAE with the RNN language model for text modeling (TextVAE). In their work, they adopt Gaussian assumption towards the VAE latent space. They show that VAE can be used for effective text modeling. Though Gaussian assumption is convenient and straightforward in mathematics, it can not guarantee good results when the underlying structure of data is complicated. Xu et al. [41] use von Mises-Fisher (vMF) distribution instead of Gaussian distribution. They show that the spherical assumption can contribute to better latent space modeling. Also, VAE models are prone to posterior collapse as point out by Bowman et al. [7], the vMF makes the KL term non-zero constant value controlled by hyperparameters, therefore suffers less from collapse issue. Besides, Dieng et al. [16] use skip connections to enforce the links between latent vectors and the likelihood function, which is also useful to avoid posterior collapse issue. Li et al. [25] argue that though previous approaches can alleviate the collapse issue, they still require careful engineering and parameter tuning. In their work, they introduce the holistic

regularisation VAE (HR-VAE) by imposing regularization towards encoder state at all time steps. This simple but effective trick can improve VAE modeling quality.

The advantage of VAE for text modeling is more diverse and controllable generation. We can easily sample from the latent space or perform interpolation between two random latent spaces to get a continuous text generation. However, sampling-based generation cannot guarantee precise text generation. A natural idea is imposing more constraints on latent space for better customized generation.

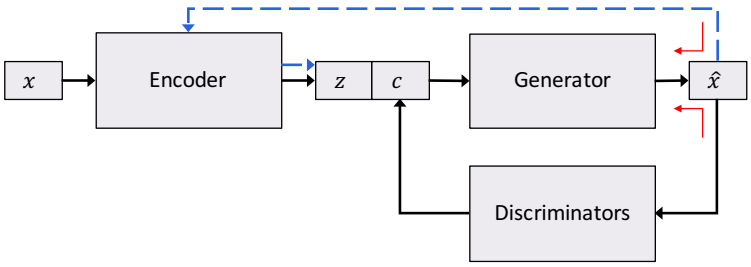


Figure 2.10: Architecture of controlled VAE generation model [20]

Hu et al. [20] proposed a controlled generation framework[3] (Figure 2.10). The framework concatenate additional structural code vector $\mathbf{c}$ with the latent vector $\mathbf{z}$. The decoder generates text based on $[\mathbf{z}; \mathbf{c}$ instead of only depend on $\mathbf{z}$ in vanilla VAE. The additional structural code vector $\mathbf{c}$ can be a discrete label or continuous vector. For example, we can use 0/1 to represent the positive/negative labels in controlled sentiment text generation. They also introduce addition discriminator, which is pretrained on existing text classification corpus. The discriminator is used to ensure the generated sentence has the same structural code $\mathbf{c}$ as its preset conditions in latent space.

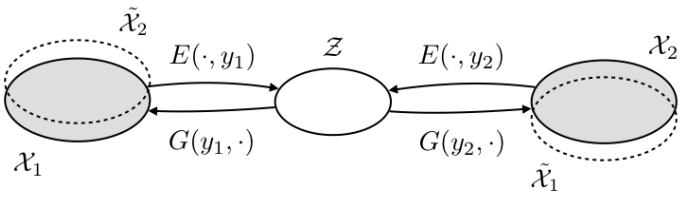


Figure 2.11: Architecture of cross-align VAE generation model [37]

[3]This figure is provided in the original paper

The framework proposed by Hu et al. [20] can work without parallel data. Shen et al. [37] further address the issue of controlled generation without parallel corpus by using the cross-alignment idea from the cycle consistency [44] in computer vision. This framework uses a style-independent encoder to map sentences into content space, then use a style-dependent decoder to reconstruct the input. This approach only performs modeling of content space and assumes the rest information is all about styles. John et al. [21] further introduced explicit modeling of latent space properties in the sentiment style transfer. They used auxiliary learning objectives to regularize the latent space for disentanglement.

All the works mentioned above focus on sentiment style transfer or some implicit text style transfer. The syntax structure is an essential factor for the language generation, but it receives little attention in recent style transfer works.

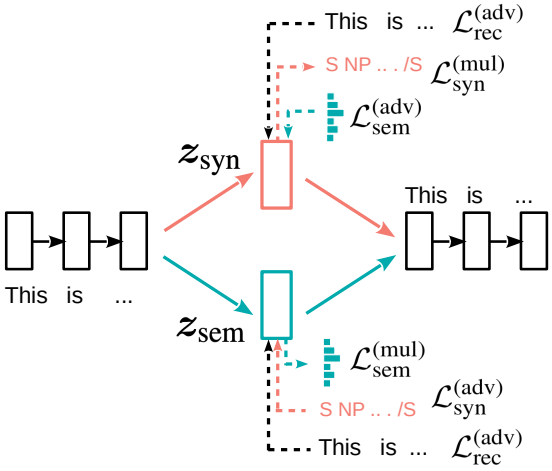


Figure 2.12: Architecture of DSS-VAE model [4]

DSS-VAE [4] is the first work in text style transfer with an emphasis on syntax transfer. Similar to John et al. [21], they adopted the multitask and adversarial regularization in the latent space. For the modeling of syntax information, they introduced the tasks of reconstructing linearized tree. In this task, they convert a sentence into a sequence of syntax labels, then reconstruct the syntax tag sequence based on latent space information. In this work, we use DSS-VAE model as one of our baseline models. The major difference between DSS-VAE and our work is the syntax modeling part. We will address this in the following chapters.

Chapter 3

Approach

In this section, we introduce the architecture of our proposed TA-VAE approach and learning objective and optimization. This chapter organizes as follows: 1) we will introduce the VAE model with disentangled syntax and content space, 2) A detail explanation about the regularization of syntax and content space is provided. 3) Last, we discuss the optimization objectives, and the techniques adopted for stabilized training of TA-VAE model.

3.1 Variational Autoencoder with disentangled space

Our model is designed based on the VAE model described in Section 2.3. Given an input document with sequence length of n, $\mathbf{x} = [x_1; x_2; ...; x_n]$, VAE will compute the probability of $\mathbf{x}$ as

$$p(\mathbf{x}) = \int p(\mathbf{z})p(\mathbf{x}|\mathbf{z})\mathrm{d}\mathbf{z} \tag{3.1}$$

In our TA-VAE setting, we assume that the latent space can be split into two parts; one part focuses on the modeling of the content information; the other part learns the syntax information. Our assumption is valid as humans can perform sentence-generation tasks for specific syntax structures or rephrase sentences to different syntax structures without changing their meaning.

Our disentangled VAE modify the single latent vector $\mathbf{z}$ into two independent latent vectors $\mathbf{z}_{\text{syntax}}$ and $\mathbf{z}_{\text{content}}$. Therefore the Eq. 3.1 can be rewrite as

$$p(\mathbf{x}) = \int p(\mathbf{z}_{\text{content}})p(\mathbf{z}_{\text{syntax}})p(\mathbf{x}|\mathbf{z}_{\text{content}}, \mathbf{z}_{\text{syntax}})\mathrm{d}\mathbf{z}_{\text{content}}\mathrm{d}\mathbf{z}_{\text{syntax}} \tag{3.2}$$

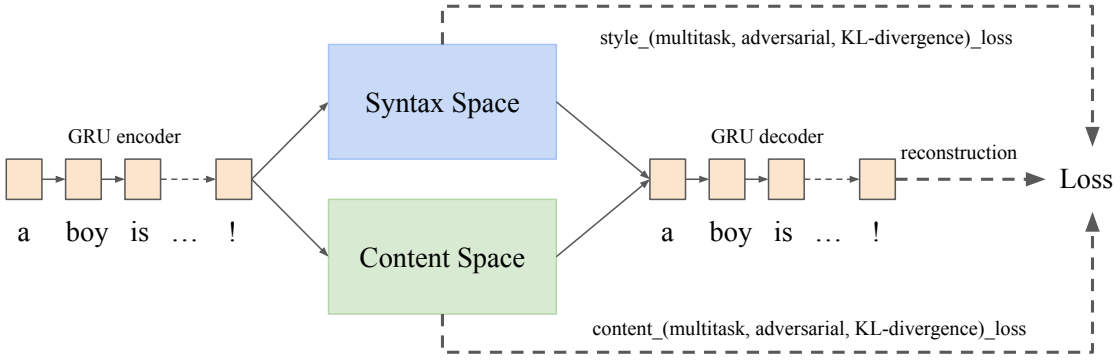


Figure 3.1: TA-VAE Model Architecture

As shown in Figure 3.1, our model first encodes the sentence with GRU encoder, then transforms the last hidden state of GRU encoder into two different latent spaces: syntax space and content space. In this work we adopt the multivariate Gaussian assumption ($\mathcal{N}(\mathbf{0}, \mathbf{I})$) towards all latent space, which is widely used in related works [7, 4]. The decoder takes the two latent vectors as input, then tries to reconstruct the input sentence. We then optimize the evidence lower bound (ELBO) during the training process.

$$\begin{aligned} \log p(\mathbf{x}) \geq & E_{q(\mathbf{z}_{\text{content}}|\mathbf{x})q(\mathbf{z}_{\text{syntax}}|\mathbf{x})}[\log p(\mathbf{x}|\mathbf{z}_{content}, \mathbf{z}_{\text{syntax}})] \\ & - KL(q(\mathbf{z}_{\text{content}}|\mathbf{x})\|p(\mathbf{z}_{\text{content}})) \\ & - KL(q(\mathbf{z}_{\text{syntax}}|\mathbf{x})\|p(\mathbf{z}_{\text{syntax}})) \end{aligned} \tag{3.3}$$

During the inference stage, we can directly sample from the disentangled spaces prior, then use the two random samples vectors as input to the decoder for text generation. We can also use the content latent vector sampled from posterior $q(\mathbf{z}_{\text{content}}|\mathbf{x})$ and syntax latent vector sampled from prior $p(\mathbf{z}_{\text{syntax}})$ for the syntax transfer task. Our disentangled latent space can enable more flexible control over text generation by change different parts of the latent information.

Besides the disentangled VAE training objective shown in Eq. 3.3, we also introduce the regularization towards the syntax and content space. We will introduce the auxiliary losses in the following sections.

3.2 Content Modeling

Modeling semantics remains a challenging problem in natural language processing. A simple assumption is that the higher the word overlap between two sentences, the higher the semantic similarity. Though there exist lots of defeats of this assumption, it is useful in lots of relevant tasks. This approach does not require any external data and can quickly adapt to different domains.

In this section, we introduce how to use the Bag-of-Word (BoW) information to measure the content preservation in latent space.

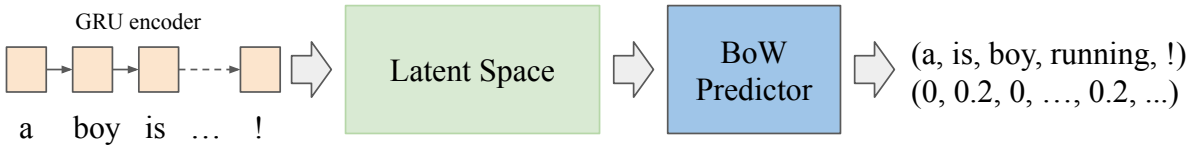


Figure 3.2: Content Preservation Prediction

The diagram is shown in Figure 3.2. We encode sentence $\mathbf{x}$ into latent vector $\mathbf{z}$ with GRU encoder. We assume a latent vector with sufficient information about $\mathbf{z}$ can predict all the words in the input sentences. This is different from reconstructing input sentences from the latent space as we do not require any sequential modeling information. The BoW target is represented as distribution, where all the entries have the probabilities sum up to one. The BoW predictor then takes $\mathbf{z}$ as input, then transforms it as the following equation as the prediction result.

In this work, we use a one-layer fully-connected neural network with softmax normalization as our BoW predictor. We can then optimize the cross-entropy between the BoW prediction and the target to enforce the information being encoded by latent space.

Previous work John et al. [21] also uses Bag-of-Word as the content modeling indicator. In their work, all non-sentiment words are excluded from the BoW prediction. In contrast, we use the full sentence as the prediction target since there is no clear definition as syntax words and content words. Experiment results show that this does not affect performance.

3.3 Syntax Modeling

In this section, we introduce supervised constituency tree composition as the syntax modeling of latent space. Our method is based on Choi et al. [13], and modify it into supervised

setting.

Unlike Section 3.2, where the last hidden state is used to calculate the latent vector and BoW prediction. The syntax modeling part needs to take the states at each time step for the step-wise tree composition. We can only use the last hidden state since it should contains sufficient information. In practice, this will lead to poor performance.

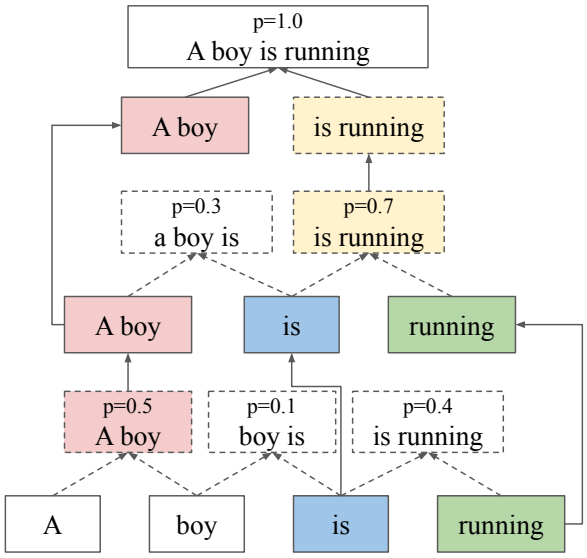


Figure 3.3: Constituency Parse Tree Composition

In step-by-step tree composition, we use each time step information instead of the last hidden state. We then use the average of each time step states for the computation of latent space statistics. As shown in Figure 3.3, for each time step, we will compute the merge probabilities between different constituents. We then merge the candidate with maximum probability and update its corresponding state. We repeat this process until all the words are merged. This process has the following three key components:

- Node Representation Transformation: how to update the corresponding node state after merging two constituents at each step.
- Parent Selection: how to choose the most likely constituents merging.
- Tree Composition: how to iterative over full sequence layer-by-layer to get a constituency tree.

3.3.1 Node Representation Transformation

As introduced in Section 2.1.2, for each time step of the input sequence, we have $\mathbf{h}_n$ denotes its corresponding hidden state.

The parent candidate is shown in Figure 3.3. For each layer (tree compose layer), we will use adjacent tokens to form candidates for the parent selection prediction. For a sequence with length n at layer i (bottom layer as layer 1), we will have $n - i$ candidates. We will do this recursively until nothing left to merge.

We use Recursive RNN as described in 2.4.2 to perform the transformation. Our computation is based on TreeLSTM (Eq. 3.6), where the parent hidden state $\mathbf{h}_p$ is computed by taking its left and right child hidden state ($\mathbf{h}_l$ and $\mathbf{h}_r$) as input.

$$\begin{bmatrix} \mathbf{i} \\ \mathbf{f}_l \\ \mathbf{f}_r \\ \mathbf{o} \\ \mathbf{g} \end{bmatrix} = \begin{bmatrix} \sigma \\ \sigma \\ \sigma \\ \sigma \\ \tanh \end{bmatrix} \left(W_{\text{comp}} \begin{bmatrix} \mathbf{h}_l \\ \mathbf{h}_r \end{bmatrix} + \mathbf{b}_{\text{comp}} \right) \tag{3.4}$$

$$\mathbf{c} = \mathbf{f}_l \odot \mathbf{c}_l + \mathbf{f}_r \odot \mathbf{c}_r + \mathbf{i} \odot \mathbf{g} \tag{3.5}$$

$$\mathbf{h}_p = \mathbf{o} \odot \tanh(\mathbf{c}) \tag{3.6}$$

3.3.2 Parent Selection

Once we have all the parent candidate at layer i, we can then perform parent selection with the representation of parents. We introduce a trainable query vector $\mathbf{q}$. This query vector can be used to measure the probability that a combination is valid.

$$\text{score}_i = \frac{\exp(\mathbf{q}\mathbf{h}_{\text{p}}^i)}{\sum_{(i)} \exp(\mathbf{q}\mathbf{h}_{\text{p}}^{(j)})} \tag{3.7}$$

3.3.3 Tree Composition

To avoid confusion of layer in the tree composition setting and the neural network setting, in this section, unless otherwise specified, all the layers refer to the tree composition stage. The layer index starts from 0, which is the initial sequence. For each merge process,

the layer index will add one, until nothing to merge. For a sequence with length n, the maximum possible tree height is $n-1$. For the last layer, where only two components left, there is no need to perform prediction. The maximum iteration number is $n-2$.

Algorithm 1: Constituency Tree Composition

```
Result: a constituency tree
initialization;
sequence_length = n;
max_iteration = n − 2;
for layer in [0, 1, ..., max_iteration] do
    candidates = propose_parent_candidate(sequence, sequence_states);
    candidate_representations = [];
    candidate_score_logits = [];
    for candidate in candidates do
        left_child, right_child = candidate;
        candidate_representation = TreeLSTM(left_child, right_child)   # Eq. 3.6;
        candidate_score_logit = query_vector * candidate_representation # Eq. 3.7;
        candidate_representations.append(candidate_representation);
        candidate_score_logits.append(candidate_score_logit);
    end
    candidate_score_probs = normalize(candidate_score_logits) # Eq. 3.7;
    merge_target_index = max_index(candidate_score_probs);
    # update the merge states and copy remaining states for next layer;
    sequence_states = update_states(merge_target_index) ;
    # update the sequence, treat the merged tokens as one token;
    sequence = update_sequence(merge_target_index) ;
end
```

During the training, we will maximize the probability at the correct position for each tree composition step. In the inference stage, the model will select the highest probability parent, then transform the two children into a single-parent node representation using Eq. 3.6.

3.4 Learning Objectives

In this section, we introduce the joint learning objectives for the TA-VAE model.

3.4.1 VAE Learning Objective

As detailed explained in Section 3.1, the optimization of VAE learning objective is equivalent to optimize the evidence lower bound (ELBO). This can be interpreted as optimization of reconstruction error and the KL distance between prior and posterior.

We minimize the following learning objective during model training,

$$\begin{aligned} \mathrm{J}_{\mathrm{vae}}(\theta_{\mathrm{encoder}}, \theta_{\mathrm{decoder}}) = & - E_{q(\mathbf{z}_{\mathrm{content}}|\mathbf{x})q(\mathbf{z}_{\mathrm{syntax}}|\mathbf{x})}[\log p(\mathbf{x}|\mathbf{z}_{content}, \mathbf{z}_{\mathrm{syntax}})] \\ & + \lambda_{\mathrm{kl\_content}} KL(q(\mathbf{z}_{\mathrm{content}}|\mathbf{x}) \| p(\mathbf{z}_{\mathrm{content}})) \\ & + \lambda_{\mathrm{kl\_syntax}} KL(q(\mathbf{z}_{\mathrm{syntax}}|\mathbf{x}) \| p(\mathbf{z}_{\mathrm{syntax}})) \end{aligned} \tag{3.8}$$

3.4.2 Multitask Learning Objectives

We use the content modeling and syntax modeling task to enforce the latent space to encode related information. As described in Section 3.2, we use the content latent space to predict the bag of word distribution $\mathbf{t}$, the content modeling objective can be written as

$$\mathrm{J}_{\mathrm{content\_multitask}}(\theta_{\mathrm{mul(c)}}) = - \sum_{w \in V} t_w \log p(w|\mathbf{z}_{content}) \tag{3.9}$$

This learning objective is similar to John et al. [21] and Bao et. al [4]. By minimizing the log probability for each BoW entry, we can encourage the content space to encode semantics related information.

For the syntax modeling of sequence with length n, given the tree composition ground truth one-hot vector $\mathbf{t}^{(i)}$ at layer i, $m_j^{(\mathrm{i})}$ denotes the merge position j in layer i. We compute the syntax modeling loss as

$$\mathrm{J}_{\mathrm{syntax\_multitask}}(\theta_{\mathrm{mul(s)}}) = - \sum_{i}^{\mathrm{num\_layers}} \sum_{j}^{n-i} t_j^{(\mathrm{i})} \log p(m_j^{(\mathrm{i})}|\mathbf{z}_{syntax}) \tag{3.10}$$

We minimize syntax multitask loss during the optimization process. It is worth noticing that the layer-by-layer composition treats each layer equally. Therefore it can emphasize more on the local structure modeling.

3.4.3 Adversarial Learning Objectives

For the adversarial learning objective, we try to remove syntax and content information from content space and syntax space, respectively. In the adversarial loss training, the gradient will not propagate to VAE related parameters. Only the auxiliary loss module parameters will be updated.

Following the same notation as in multitask loss computation, we have the adversarial objectives as follows.

$$\mathrm{J}_{\text{content\_adv}}(\theta_{\text{adv(c)}}) = -\sum_{w \in V} t_w \log p(w|\mathbf{z}_{syntax}) \tag{3.11}$$

$$\mathrm{J}_{\text{syntax\_adv}}(\theta_{\text{adv(s)}}) = -\sum_{i}^{\text{num\_layers}} \sum_{j}^{n-i} t_j^{(\text{i})} \log p(m_j^{(\text{i})}|\mathbf{z}_{content}) \tag{3.12}$$

By maximizing the content adversarial loss 3.11, the syntax space will be unable to predict the BoW distribution; therefore, it can be used for the removal of content information from syntax space. Similarly, the content space will fail for the tree composition task by optimizing syntax adversarial loss 3.12. This can serve as the removal of syntax information from content space.

3.4.4 Optimization

The optimization process has three stages. The first stage is to minimize the syntax adversarial loss for the constituency tree predictor parameters $\theta_{\text{adv(s)}}$. The second stage is to minimize the content adversarial loss for the bag-of-word predictor parameters $\theta_{\text{adv(c)}}$. Last, we minimize the joint loss J_{overall} with respect to all the parameters of TA-VAE model except the adversarial modules. The joint objective for last stage optimization is defined as following equation.

$$J_{\text{overall}} = J_{\text{VAE}} + J_{\text{syntax\_multitask}} + J_{\text{content\_multitask}} - J_{\text{syntax\_adv}} - J_{\text{content\_adv}} \tag{3.13}$$

For each minibatch of training data, we alternatively optimize the three learning ob-

jectives described in Algorithm 2.

Algorithm 2: Optimization Process

foreach *mini-batch* **do**
  minimize $J_{\text{syntax\_adv}}(\theta_{\text{adv(s)}})$ w.r.t $\theta_{\text{adv(s)}}$
  minimize $J_{\text{content\_adv}}(\theta_{\text{adv(c)}})$ w.r.t $\theta_{\text{adv(c)}}$
  minimize J_{overall} w.r.t $\theta_{\text{encoder}}, \theta_{\text{decoder}}, \theta_{\text{mul(s)}}, \theta_{\text{mul(c)}}$
end

We use the Adam Optimizer [22] for the joint optimization. We also use the KL-anneal trick and word dropout trick introduced by Bowman et al. [7] to stabilize the training process.

VAE training often suffers from posterior collapse issue. If we directly train the learning objectives, the KL loss will become zero very quickly. Under the circumstance, the VAE model will degenerate into an RNN language model. The KL-anneal strategy sets the KL weight to zero initially and gradually increases it during the training. Likewise, the word dropout also alleviates this issue by randomly replacing the input sentence with special tokens. The two tricks are helpful in text-based VAE models, as confirmed by lots of related works [7, 3, 21, 4].

Chapter 4

Experiments

In this chapter, we will test the performance of our model under different settings and compare it with baseline models. This chapter organizes as follows: we will first evaluate the generation quality of our models using the standard language model benchmark dataset. We will test our TA-VAE model's performance on syntax transfer and its related tasks. We will also provide quantitative and qualitative analysis of the experiment result.

4.1 Dataset

Penn Treebank (PTB) [29] is a dataset constructed with stories from Wall Street Journal (WSJ). It is a widely used dataset for benchmark language model performance. We preprocess the dataset following the common procedures described by Mikolov et al. [31] to reduce the total vocabulary size to 10,000 tokens. The vocabulary is constructed with tokens frequency, for tokens not in vocabulary, we replace it with a special token. The dataset contains 42068/3370/3761 sentences for training, validation, and test, respectively. We also use the coreNLP [28] constituency tree parser to provide annotation for the training of our model TA-VAE.

Stanford Natural Language Inference (SNLI) [8] is a large-scale natural language inference dataset build with crowd-sourced workers. The workers are required to write English caption for images. Though SNLI is designed for natural language inference task, the dataset is suitable for language modeling tasks as the structure is much cleaner than other web-crawled datasets. The original SNLI dataset has 550,152/10,000/10,000 train/validation/test pairs. We convert the sentence pairs into single sentences, result in

628,498/13,133/13,131 train/validation/test pairs. Unless otherwise specified, we will use this single-sentence version SNLI for all the experiments in this work.

4.2 Models

We include the following methods for comparison to our approach.

- Deterministic AE (DAE): this is the basic version of autoencoder without any regularization of the latent space.
- Text VAE [7] is the vanilla VAE model with Gaussian prior towards the latent space. It also uses various tricks (word dropout and KL weight annealing) for training.
- DSS-VAE [4] is the first syntax transfer model using linearized tree for syntax modeling.
- TA-VAE (Ours) is our proposed model with constituency tree and content modeling for latent space regularization.

4.3 Text Reconstruction

Text reconstruction task aims to generate the same text as the input. This task can measure the model's capability to capture information into the latent space. This is very important for the syntax style transfer related downstream tasks, as the capability to capture information in latent space will affect the precision of controlled generation performance.

4.3.1 Evaluation Metrics

Bi-Lingual Evaluation Understudy (BLEU) [33] is an evaluation method in machine translation, which is used for the measurement of word overlap between generated text and gold standard. There are multiple variants of BLEU, named after BLEU-N, where the N denoted N-gram. For example, BLEU-1 measures the overlap in word level, while BLEU-3 calculates using trigram overlap. The higher the BLEU score, the more overlap between the generation and golden reference. In most cases, the BLEU score is positively correlated with model quality though it only cares about overlap regardless of coherence/factual correctness. We will report the BLEU-1,2,3,4 as well as the average score to provide the reader with a better understanding of model performance.

4.3.2 Experiment

For models with disentangled latent space (DSS-VAE, TA-VAE), we use the mean of gaussian distribution in syntax and content as the latent representation, respectively. We measure the reconstruction performance with the word-overlap, using the BLEU scores and its variant. The higher the BLEU score, the better the reconstruction performance.

parameter	value
vocabulary size	20000
max sequence length	20
embedding size	128
encoder hidden size	128
decoder hidden size	128
latent size	64
word dropout rate	0.5
KL_weight	1.0

Models	BLEU↑				
	1	2	3	4	Average
Ground Truth	100	100	100	100	100
DAE	83.08	75.45	70.22	65.54	73.57
VAE	57.73	43.64	35.31	28.47	41.29
DSS-VAE	63.96	50.99	42.83	35.93	48.43
TA-VAE	67.96	54.69	45.81	38.45	51.73

Table 4.1: Left: Hyperparameter settings for SNLI reconstruction task; Right: reconstruction Performance on SNLI dataset

parameter	value
vocabulary size	10000
max sequence length	50
embedding size	128
encoder hidden size	256
decoder hidden size	256
latent size	128
word dropout rate	0.5
KL_weight	1.0

Models	BLEU↑				
	1	2	3	4	Average
Ground Truth	100	100	100	100	100
DAE	53.00	38.23	29.32	23.09	35.91
VAE	39.90	25.40	17.55	12.78	23.91
DSS-VAE	40.34	25.61	17.66	12.77	24.09
TA-VAE	40.82	25.80	17.87	13.08	24.39

Table 4.2: Left: Hyperparameter settings for PTB reconstruction task; Right: reconstruction Performance on PTB dataset

We use the SNLI and PTB[1] datasets for this reconstruction task. SNLI dataset is much simple and clean in comparison to PTB dataset, therefore we adopt different parameter groups (left part of Table 4.1 and Table 4.2) for the two dataset to address its difficulties for language modeling following the settings of [3, 7, 4]. VAE is prone to have KL term collapse (posterior collapse), where the model will degenerate to a pure RNN language

[1]Due to the copyright issues, we use the free-available version with a total vocabulary size 10,000

Model	Original	Reconstructed
TA-VAE	the man is drunk	the man is getting drunk .
DSS-VAE	the man is drunk	the man is hot
TA-VAE	a little girl is enjoying cake outside .	a little girl is enjoying a birthday party .
DSS-VAE	a little girl is enjoying cake outside .	a little girl is eating hot tub .

Table 4.3: Examples of Reconstructed Sentences on SNLI dataset

model without relying on the latent vector. To avoid this issue, we use the KL annealing approach [3, 7] by gradually increase the KL term weight from zero to fixed value during the training stage. We also use word dropout by random replace words into special tokens (UNK). This can also alleviate the KL term collapse by forcing the model to rely on latent representation, as suggested by Bowman et al. [7].

The DAE gets the best reconstruction performance among all the models. This result is within our expectation as the deterministic model can learn an RNN language model without additional constraint. Therefore DAE is much easier to train than other variational models.

We also observe that the gap of performance between syntax-based models and vanilla VAE varies from different datasets. The SNLI dataset is much easier in terms of content information, and the main bottleneck is syntax structure modeling. While PTB dataset is much more diverse in terms of semantics, syntax and content modeling are both crucial. This observation is confirmed by the absolute BELU score difference of SNLI and PTB dataset (average 28.47 versus 12.78).

Overall, we can conclude that models with explicit syntax structure modeling (DSS-VAE, TA-VAE) are better than those without it. Our proposed model is better than DSS-VAE in terms of BLEU scores (3.3 absolute BELU score improvement) on SNLI dataset. This suggests modeling the full tree information can get better performance in comparison with the linearized tree. The increase is not statistically significant on PTB dataset, as the bottleneck mentioned above is content modeling.

Though content modeling is crucial in language generation, the emphasis is mainly on syntax structure modeling. An ideal dataset for this task should have semantics within the capabilities of small-sized language models and sufficient diversity in syntax structure. Therefore SNLI dataset is much proper than PTB in terms of syntax related modeling. We will mainly use SNLI for the major part of this work.

4.4 Unconditional Generation

In this section, we will run unconditional generation experiments. The unconditional generation task aims to generate sentences by sampling from the prior distribution of latent space. The generation quality of the sentences and their diversity can be used as metrics to evaluate the quality of latent space modeling. Based on the conclusion from Section 4.3, we use SNLI dataset with more emphasis on the syntax structure.

4.4.1 Evaluation Metrics

Forward Language Model Perplexity (F-PPL) [43] is used to measure the generative model's capability to synthesize data. F-PPL uses a language model trained on external datasets, then uses this pretrained language model to evaluate synthetic samples. In our settings, the synthetic texts are generated by sampling the latent space of models. The F-PPL score reflects the fluency of the synthetic texts. The higher F-PPL scores, the better the model.

Reverse Language Model Perplexity (Rev-PPL) [43] aims to measure the quality of the generated sentences. We will first generate 100,000 sentences by sampling from the latent space (both syntax and content space), then use it to train a language model. The language model is used to evaluate the perplexity of the training set. This reverse evaluation stage can measures the quality, diversity of the generation model. For example, a model with mode collapse may get a good score from the forward language model perplexity by collapse to a specific state. In contrast, it will get an extremely low score on reverse language model perplexity. Therefore, the higher the Rev-PPL score, the better the generation model.

Average Length (Avg. Len) calculates the average of all generated sentences' length. This metric can provide a better understanding of the similarity between the training set and the unconditional generated sentences. We should anticipate a perfect generation model will have the same average length as its training data.

N-gram Diversity (Avg Diversity) [3] is introduced to quantify the mode collapse issue. Similar to the evaluation of Rev-PPL, we first sample 100,000 sentences from the latent space. Then calculate the ratio of unique N-grams using equation 4.1. A well-trained model will have relatively high diversity, while a collapsed model (or deterministic model) will have low (or zero) diversity. We will use the average of (3,4,5)-grams for the evaluation.

$$\text{N-gram diversity} = \frac{\text{\# of unique n-grams}}{\text{\# of n-grams}} \tag{4.1}$$

4.4.2 Experiment

The unconditional generation task aims to generate sentences by sampling from the prior distribution of latent space. The generation quality of the sentences, as well as its diversity, can be used as metrics to evaluate the quality of latent space modeling.

We will use the forward language model perplexity (F-PPL) to evaluate the coherence of the generated text and reverse language model perplexity (Rev-PPL) to measure the similarity between generation and the original data domain. We also provide the average length and average diversity to provide better quantify of the generation quality of different approaches. Besides the baseline models, we also include the training data as gold reference to give a baseline of different metrics.

#	Models	F-PPL↓	Rev-PPL↓	Avg. Len	Avg. Diversity↑
1	Gold Reference	112.93	-	9.75	0.39
2	VAE	129.12	20.51	8.55	0.49
3	DSS-VAE	119.13	18.80	8.56	0.56
4	TA-VAE	92.45	18.51	8.44	0.5

Table 4.4: Unconditional generation quality of different approaches

We use a language model trained on external data to evaluate the forward language model perplexity (F-PPL). In this work, similar to Bao et al. [4], we use the one billion word benchmark [10] as an external dataset. According to Table 4.4, system with explicit modeling syntax information can get better F-PPL scores. TA-VAE gets the best performance. To our surprise, the TA-VAE model even gets a better F-PPL score than the gold reference. Since there exist some rare sentences from gold reference, the distribution mismatch between them and the external dataset may contribute to this observation. In this case, the forward perplexity of gold reference is under-estimate. While for the TA-VAE model, it models the most common semantics in the training set. Therefore TA-VAE output resembles the external dataset and gets a better F-PPL score. Though the forward perplexity has flaws, we can still conclude that TA-VAE can generate more coherent sentences than other models.

The reverse language model perplexity (Rev-PPL) measures the distance between unconditional generated sentences and gold reference. For all the models, we sample 10,000

sentences from the latent space, then train a language model using KenLM [18]. We then measure the negative log-likelihood of the training set based on the estimate of the language model mentioned above. A lower reverse language model perplexity score indicates the higher similarity between synthetic sentences and training set.

Our model can show best performance as shown in Table 4.4. The reverse language can also be used as an indirect measurement of the mode collapse issue of VAE models. A collapsed model will generate the same output during the unconditional generation and will have relatively high Rev-PPL scores. Like the forward language model perplexity, the reverse language model perplexity may also be underestimated due to the lack of representation power of the third-party language model. It is almost impossible to get zero log-likelihood on the training set. Therefore, the theoretical lower bound of reverse perplexity is 0, while we can never get that in practice. The average length of different models is similar, while still has one token (on average) difference with the gold reference. Nevertheless, we can still conclude that our TA-VAE model's generated sentences have a relatively high similarity with gold reference.

The mode collapse issue is prevalent in the training of variational autoencoders. It is important to measure the diversity of the generated sentences. We compute the unique n-grams from the sampled 10,000 sentences. We compute the average of 3,4,5 grams diversity scores. From the Table 4.4, DSS-VAE get the best diversity score. However, given the low coherence measured by the F-PPL score, we can conclude that the incoherence generation results in a diverse generation. This assumption is supported by the gold reference's diversity score (0.39 versus 0.56). Based on the diversity scores, we can conclude that all the models don't suffer too much from mode collapse issue, as a mode collapsed model will have close to zero diversity score.

To summarize, we can conclude that our TA-VAE model gets the best performance compared to other baseline models. Our TA-VAE model has a good quality of latent space modeling.

The weight of KL terms directly contributed to the latent space modeling of VAE based models. We perform generation quality analysis by using different KL weight to regularize the latent space. Different from Table 4.4, where the result is based on optimal result from different models. In this analysis, we use the average of different parameter groups (e.g., different weight towards content modeling and syntax modeling) under the same KL weight. This setting can show overall performance under different hyperparameter settings. It is more reliable than the evaluation based on a single model output.

As shown in Table 4.5, by increasing the KL penalty, we can get better fluency in terms of unconditional generation. This observation is consistent with previous studies [4, 7].

KL_weight	F-PPL↓	Rev-PPL↓	Sentence Diversity↑	Avg. Diversity↑
1.00	53.13	17.36	0.70	0.13
0.75	75.59	16.57	0.85	0.24
0.50	109.32	16.30	0.92	0.35
0.25	159.65	16.43	0.96	0.48

Table 4.5: Unconditional generation quality of TA-VAE under different KL weight

However, we find that the diversity of generated sentences is much worse than the optimal results shown in Table 4.4. This observation indicates a large portion of the parameter groups is still suffering from mode collapse issue. The lack of diversity of issues makes this average score not directly comparable with the above-mentioned optimal result. The overall trend is still valid. If we want to get a better latent space quality, we should increase the KL weight. If we need to preserve more content in reconstruction or controlled generation, we need to decrease the KL weight. A crucial part of this work is to find the balance between all these parts. We will further discuss this in ablation studies.

4.4.3 Case Analysis

In the previous sections, we conduct multiple experiments under the settings of general VAE generation. We show that our model with explicit content and syntax modeling, can outperform previous models. This section will perform more detailed case studies of the syntax and content space, respectively.

Sample-based generation syntax and content spaces

For the left part of Table 4.6, we first sample an random vector from Gaussian distribution (mean=0, std=1) as the fixed content vector. We then generate multiple vectors follow the same distribution as the syntax representation, then concatenate the content and syntax vector for the TA-VAE model generation. Likewise, for the right part, we fix the syntax vector and sample multiple content vectors.

According to the first example of the sampling syntax space, we can conclude the fixed content vector corresponds to "child, watch, movie, play" as all the sentences have these keywords. We can also find that syntax structure is quite diverse, given the same content vector. The syntax template ranges from "someone is doing" to "there is someone doing". Similar things can also be observed in the second example, where the content

Sample Syntax Space	Sample Content Space
a child is watching a monument play	there is towing towing passing by the street
a child watching a movie	there is a group laying around people nearby
a child is watching a monument	men are welding while standing outside
there are a canine watching	there are girls <unk>
a child is watching a fire stand	there is <unk> taking place
there is a child watching the riverbank	there is a lady pumping up in the air
a child is watching a cactus	snowboarder is skiing
the child looks at a historic area	there are other musicians talking
a black dog laying on a couch outside	the women are dancing
there is a man laying on a couch	the man is inside reading
a man is laying on the couch	the man is cooking dinner
a man laying on the couch with a dog	the woman is standing inside
a dog is sleeping	the women are going crabbing
a dog is laying on the couch	the couple are carrying groceries
a dog is on the couch	a man is cutting fruit
a man on a couch	a man is doing yoga
a dog is laying on a couch outside	a boy is playing video games

Table 4.6: Generation Examples of from syntax and content space

vector corresponds to "dog, laying, man, couch". To conclude, we can find sentences with roughly the same content with different syntax template. These examples show the excellent quality of the TA-VAE model's syntax space.

Similarly, for the first example of sampling content space, we can observe that the "there be doing" template dominates the generation with different content information. The second example corresponds to "someone is doing something".

We find that the syntax template is more robust than the content template. For the left part of Table 4.6, the content sometimes shift a lot given different syntax template. While for the right part of Table 4.6, the syntax is almost fixed, no matter how we change the content information.

Interpolation of syntax and content spaces

We further perform interpolation in the latent space to better illustrate the disentangled syntax and content space. We will draw a randomly sampled vector as the fixed content

Interpolate Syntax Space	**Interpolate Content Space**
a man is riding on a bike .	a boy running in the hot air
a man is riding his bike down	a dog running in the hot air
a man is riding a bike ride	a boy sitting in the hot air
a man riding to a red bike	a boy sitting in the air with drinks
a man riding his bike	a girl sitting in the air with drinks
a man riding a bike in asia	a girl sitting in the street watching tv
man riding his bike in a parade	a girl sitting in the park watching tv
person riding his bike in a parade	a girl sitting in the park watching a movie
man riding a bike in the street	a girl sitting in the park at a restaurant
there is a chihuahua and dog lying on the floor	there is a path
there is a chihuahua and dog lying on the floor	there is a picnic path
the dog on a leash is lying on a bed	there is a man sleeping
the dog biting a muzzle on a yellow dog nearby	there is a man sleeping alone
the dog is on a leash and a muzzle nearby	there is a man at the beach
the dog is on a leash and muzzle nearby	there is a man in the street
the dog is biting a dog on a yellow floor	there is a man unk alone
the dog is biting on a leash on the floor	there is a man unk the night
the dog is lying on a rug on the floor	there is a man is singing at the local club

Table 4.7: Interpolation examples of from syntax and content space

information for the interpolation in the syntax space, then concatenate with linear interpolated syntax vector as the latent vector. Similarly, for the content interpolation setting, we only manipulate the content vector while keeps the syntax vector fixed.

As shown in left part of Table 4.7, the syntax space can generate different sentences using different syntax template. The first syntax interpolation example shows the changes from "someone is doing something" to "someone doing something". The second example shows the changes from "there is something doing" to "something is doing". Though some duplication and incoherent generation exist (e.g., duplication on the second example), we can observe a continuous change of syntax structure (e.g., local structure change) during interpolation in most cases.

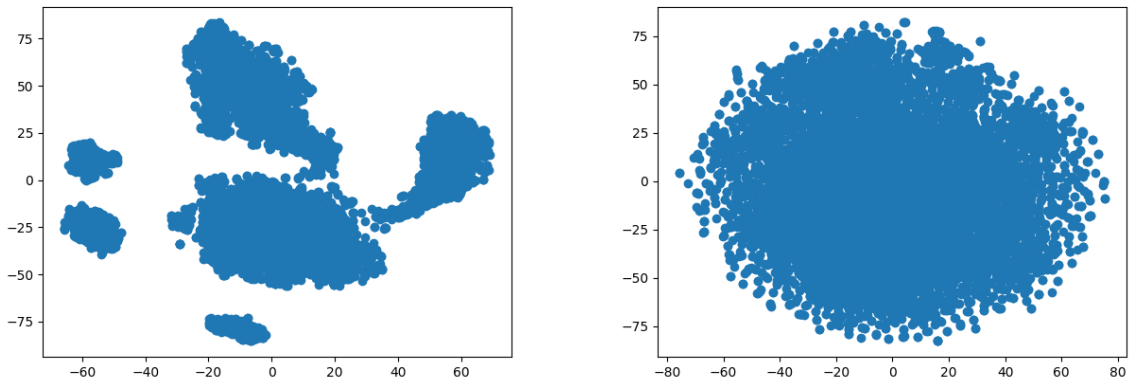
In the interpolation of content space, we can observe the syntax template almost fixed no matter how content information changes. The first example shows the content changes from "boy running" to "girl sit in the park at resturant" while keeps the syntax ("someone doing") the same.

In conclusion, the interpolation experiment shows that our disentangled spaces can capture the necessary information about syntax and content, respectively.

4.4.4 Visualization of Disentangled Space

In this section, we perform the visualization of the latent space for a better understanding of the properties of the disentangled space. By visualize the syntax space and align it with corresponding samples, we can easily classify the syntax template of each cluster.

For each sentence in the SNLI dataset, we obtained its corresponding representation for syntax and content spaces, respectively. Then reduce the data to two dimensions using t-SNE [27].



(a) Syntax Latent Space

(b) Content Latent Space

Figure 4.1: Visualization of Disentangled Latent Space on SNLI dataset

We can observe that the syntax space has six distinct clusters while can hardly identify any groups in content space. This is within our expectation, as the content information of SNLI dataset varies a lot. In contrast, the syntax structure of SNLI is much cleaner, with some significant structures (e.g. there be, someone doing, someone is doing, etc).

4.5 Text Syntax Transfer

In this section, we propose text syntax transfer task. This task's objective is to change the syntax structure of sentences while keeping content information the same. This task is a good testbed of the disentanglement of syntax and content. We test multiple models on this task using the SNLI dataset.

4.5.1 Evaluation Metrics

Tree Edit Distance (TED) measures the minimal operations to change a source tree to a target tree. It measures the distance between trees. All the operations are organized under the node operation setting, including insert, delete, and rename. In this work, we use the open-source implementation[2] based on the algorithm described in Zhang et al. [42].

Human Ranking Score performs a qualitative measurement of the syntax transfer tasks. We invite two graduate students as human evaluators. Evaluators are required to provide a ranking of VAE, DSS-VAE, and TA-VAE models' output based on the syntax transfer quality. Each model will get a score ranging from 1 to 3 to indicate generation quality. The higher the score, the better the generation quality.

Human Ranking Consistency measures the agreement between human evaluators. We use the Spearman correlation coefficient to quantify the consistency.

Bi-Lingual Evaluation Understudy (BLEU) [33] described in Section 4.3.1 is used to measure the word overlap between the syntax-transferred sentence and the original sentences. This metric can quantify content preservation during the syntax transfer. Though BLEU may be a biased estimation for content preservation, in most cases, it is still reliable and widely used in lots of language generation evaluation tasks.

4.5.2 Evaluation Dataset

Since there is no existing parallel evaluation set for syntax transfer, we create validation/evaluation sets by random selection from the SNLI validation/test sets. For the quantitative assessment (BLEU, TED score), we create 1000 sentence pairs by randomly sampled from the SNLI test set as our syntax transfer evaluation. For the qualitative assessment (human ranking score), we randomly select 25 sentence pairs from the output of VAE, DSS-VAE, and TA-VAE in the evaluation test set.

4.5.3 Syntax Transfer Experiment

In this section, we compare VAE and DSS-VAE with our models. Our experiment settings are provided in Table 4.8. The VAE model parameter is the same as all the experiments mentioned above. For the DSS-VAE model, we use the optimal parameter group provided by Bao et al. [4] in their paper appendix. For our TA-VAE model, the optimal parameter is

[2] https://github.com/timtadh/zhang-shasha

selected using the validation performance. For the other common parameters (e.g., dropout probability), we use the same settings as described in Table 4.1.

Parameters	VAE	DSS-VAE	TA-VAE
latent size	128	128	128
reconstruction weight	1.0	1.0	1.0
syntax KL weight	1.0	0.66	0.5
content KL weight	1.0	0.33	0.5
syntax multitask weight	0.0	1.0	1.0
content multitask weight	0.0	5.0	1.0
syntax adversarial weight	0.0	0.5	1.0
content adversarial weight	0.0	0.5	0.05

Table 4.8: Parameter settings for different models

For the syntax transfer experiment, we measure the word overlap and the tree edit distance between different sources, including content template (ConT), syntax template (SynT), transferred (Trans) generation, and reconstruction (Rec). We use the following notations (pairs) in the experiment result Table 4.9 and Table 4.10:

- Trans-ConT: the transferred sentence and its corresponding content template.
- Rec-ConT: the reconstruction of content template and its corresponding content template.
- ConT-delta: the absolute difference between the Trans-ConT and Rec-ConT. This term measures the relative change under different metrics.
- Trans-SynT: the transferred sentence and its corresponding syntax template.
- Rec-SynT: the reconstruction of content template and syntax template. This term provides the baseline performance of how much overlap between the content template and syntax template.
- SynT-delta: the absolute difference between the Trans-SynT and Rec-SynT. This term measures the relative metric change between the transferred sentences and content template with respect to the syntax template.

In Table 4.9 and Table 4.10, $\uparrow$ denotes the higher score, the better the performance and $\downarrow$ denotes the lower score, the better performance.

#	Model	Trans-ConT↑	Rec-ConT	ΔConT	Trans-SynT↓	Rec-SynT	ΔSynT
1	VAE	28.81	38.93	10.12	6.72	5.54	1.18
2	DSSVAE	17.08	48.82	31.74	10.06	6.01	4.05
3	TA-VAE	**47.34**	68.3	20.96	6.76	6.11	0.65

Table 4.9: Syntax Transfer BLEU score

#	Model	Trans-ConT↑	Rec-ConT	ΔConT	Trans-SynT↓	Rec-SynT	ΔSynT
1	VAE	60.91	49.74	11.17	152.07	153.87	1.80
2	DSSVAE	137.06	37.49	99.57	143.48	152.71	9.23
3	TA-VAE	64.37	29.58	34.79	**138.88**	149.92	11.04

Table 4.10: Syntax Transfer Tree Edit Distance

For the syntax transfer task, we hope the transferred sentences maintain a high level of word overlap with the content template and get closer syntax distance (tree edit distance) with the syntax template. Under these criteria, we can conclude that our TA-VAE model has the best performance. We can observe that our model can maintain the highest word overlap between the transferred sentence and content template (Trans-ConT in Table 4.9) and the lowest TED score between the transferred sentence and syntax template (Trans-SynT in Table 4.10). Our model can reduce the TED score by 11.04 absolute points, which is non-trivial under the settings of syntax distance.

In contrast, DSS-VAE model has a similar syntax distance score compared to TA-VAE (TED score 143.48 versus 138.88) but with relatively low content preservation performance. According to Table 4.9, the DSS-VAE model has the lowest BLEU score between the transferred sentence and content template (Trans-ConT). As shown in Table 4.10, the tree distance between the transferred sentence and the content template (Trans-ConT) is much larger than that of TA-VAE model, this indicates the change of syntax behavior affect the overall generation performance. The disentanglement of syntax and content in DSS-VAE and the syntax modeling is not as good as our model.

We also include the vanilla VAE model as a baseline. We can observe the vanilla VAE model has relatively poor performance for the syntax transfer task. The TED score between the transferred sentence and the syntax template is small and only provides a 1.80 absolute point change. The BLEU score between the transferred sentence and the content template changes a lot (10.12 absolute point). VAE model can only learn to change the sentence in token level, while failed to modify it in syntax structure level. This observation suggests the disentangled modeling is essential, especially for the syntax transfer task.

To conclude, 1) the disentanglement of syntax and content space is essential for the

syntax transfer task (VAE versus DSS-VAE and TA-VAE). 2) our model with hierarchical modeling of syntax structure can get better syntax modeling performance than DSS-VAE, which used the linearized syntax representation.

4.5.4 TA-VAE Ablation Analysis

In this section, we conduct the ablation study of TA-VAE training. We analyze the effect of different learning objectives on the performance of the syntax transfer task. Our ablation study is built on top of the optimal hyper-parameter group and result in Section 4.5. In all the experiments, we keep the reconstruction objective and the KL objective unchanged, as these two objectives have been discussed in Section 4.3 and Section 4.4.

We evaluate the performance of models with different learning objectives (multitask and adversarial objectives in syntax and content spaces) using the BLEU score and tree edit distance as Section 4.5. For simplicity, we use the notation mul to represent the multitask learning objective, and "s/c" prefix to denote the learning objectives variant in syntax and content space, respectively.

#	Model	Trans-ConT↑	Rec-ConT	ΔConT	Trans-SynT↓	Rec-SynT	ΔSynT
1	sc_mul	27.83	69.59	41.76	13.86	6.04	7.82
2	sc_mul+cadv	42.65	69.39	26.74	8.97	5.91	3.06
3	sc_mul+sadv	25.11	69.06	43.95	15.23	5.95	9.28
4	smul+sc_adv	44.96	65.31	20.35	6.78	6.16	0.62
5	cmul+sc_adv	47.34	68.74	21.40	6.92	6.18	0.74
6	TA-VAE	**47.34**	68.3	20.96	6.76	6.11	0.65

Table 4.11: Ablation study of Syntax Transfer BLEU score

#	Model	Trans-ConT↑	Rec-ConT	ΔConT	Trans-SynT↓	Rec-SynT	ΔSynT
1	sc_mul	125.26	29.04	96.22	107.62	150.96	43.34
2	sc_mul+cadv	101.14	28.72	72.42	138.27	150.69	12.42
3	sc_mul+sadv	131.63	29.42	102.21	111.91	151.12	39.21
4	smul+sc_adv	57.09	28.24	28.85	146.52	151.73	5.21
5	cmul+sc_adv	64.13	29.57	34.56	143.19	149.76	6.57
6	TA-VAE	**64.37**	29.58	34.79	138.88	149.92	11.04

Table 4.12: Ablation study of Syntax Transfer Tree Edit Distance

According to the row #1 and #6 in Table 4.11 and Table 4.12, we can conclude that the adversarial objective is crucial for the success of syntax transfer. The row #1 has one

of the least word-level overlaps with the content template (Trans-ConT). Once we add the content adversarial objective (row #2) on top of row#1, we observe a huge improvement of the word overlap with the content template. This is within our expectation as the content adversarial objective can remove the content information from the syntax space. Therefore the syntax template has much less interference over the content information. If we add the syntax adversarial objective (row #3), we can observe smaller tree edit distance to the syntax template according to Table 4.12. The syntax template dominates the generation as the syntax adversarial objective removes syntax information from the content space. We also observe that row #3 has one of the worst word overlap (Trans-ConT BLEU score) with the content template. Because solely adopt syntax adversarial objective cannot remove content information from syntax space.

Besides, we explore the importance of different multitask objectives. We find that row #5 is better than row #4 in terms of BLEU score with explicit modeling of content information (Rec-ConT 68.74 versus 65.31 BLEU score). We also observe that there is no distinct difference between row #4 and row #5. The significant statistics changes in the two tables come from the effect of different adversarial objectives.

According to the ablation study, the key to making a successful syntax transfer is the trade-off between the adversarial objectives.

4.5.5 Case Analysis of Syntax Transfer

In this section, we first introduce the human evaluation of the transfer syntax performance. Then a detailed case analysis is provided.

Human evaluation is a standard metric in language generation-related tasks, though it's more subjective than TED and BLEU scores. It can provide a better understanding of the generation quality. An excellent automatic evaluation score doesn't guarantee better performance as there is a mismatch between the evaluation metrics and human judgment. The BLEU score and TED score are indirect measurements of the syntax transfer task. Therefore we adopt human evaluation to get a better understanding of the transfer quality.

To reduce the subjective issue of human evaluation, we also use the consistency measurement (Human Ranking Consistency in Section 4.5.1) to quantify the agreement between different human evaluators. Due to the time-consuming of the human evaluation, we only use a subset of the evaluation set, as described in Section 4.5.2.

According to the human evaluation result, the TA-VAE model has the best performance. Based on human evaluators' feedback, the DSS-VAE model can perform excellent syntax

Model	Ranking Score↑	Ranking Consistency↑
VAE	1.44	0.380
DSS-VAE	2.02	0.468
TA-VAE	**2.54**	**0.564**

Table 4.13: Human Evaluation on SNLI syntax transfer

templated-based transfer but tend to change the original words into different words with similar meanings. The VAE model behaves like a random generation. Therefore, it has the worst scores.

We also randomly select some examples from the evaluation set in Table 4.14. We have the following observations.

- **Content drift:** Though TA-VAE suffers from the content drift issue though it can maintain the most critical content information. In the fifth example, the TA-VAE model capture the "woman, carrying", while fails to generate the "child", instead replace it with something. Likewise, the DSS-VAE also has this issue by adding non-exist content information "outdoor" in this example. This is caused by the gap in the mismatch between distributed representation and the discrete token output.
- **Incompatible Syntax Transfer:** The syntax template is not necessarily compatible with the content. There is no natural way to generate a proper transferred sentence with the syntax template in the fifth example. We can observe all the models fails to behave appropriately under this setting.
- **Length Bias:** The DSS-VAE model tends to encode the length of the syntax template as part of the syntax information. This phenomenon results from the auto-regressive generation of linearized parse trees. This generative syntactic structure modeling objective will introduce length bias into the latent space. Our model doesn't exist this issue as our hierarchical tree composition emphasizes more on the local tree structure regardless of length.

Syntax Transfer Examples
VAE: a young boy wearing a helmet is about life **DSS-VAE:** a skateboarder is wearing a helmet helmet . **TA-VAE:** a boy is wearing boots and a helmet . **Content Template:** a young boy wears a helmet and biking boots . **Syntax Template:** the person is surfing on the water .
VAE: a man in black holds a baby . **DSS-VAE:** a man is putting a hat on a video game . **TA-VAE:** a man is sitting in a basement . **Content Template:** a man in glasses in a basement . **Syntax Template:** a woman is talking on the phone in her bedroom .
VAE: there is a model outside . **DSS-VAE:** a man moving an object near a . **TA-VAE:** the animal is outside outside . **Content Template:** there is an animal outside . **Syntax Template:** a woman vendor gazes into the distance .
VAE: woman got a telescope **DSS-VAE:** A woman chases a child outdoors . **TA-VAE:** a woman carrying something **content template:** woman carrying a child **syntax template:** a woman runs down the field .
VAE: the woman is dancing with her dog . **DSS-VAE:** People were seated at the tables , the other is in the same room . **TA-VAE:** there is a woman at home . **content template:** the woman is yelling at the package . **syntax template:** while two students are jumping , others are sitting or standing .

Table 4.14: Syntax Transfer Examples of VAE, DSS-VAE and TA-VAE

Chapter 5

Conclusion and Future Work

In this work, we explore the controlled text generation with an emphasis on modeling syntax information. Our proposed model, TA-VAE, can do flexible modification over different syntax templates. We also perform ablation study and case analysis for quantitative and qualitative analysis of various regularization over latent space. We show that our hierarchical tree composition for the syntax modeling outperforms the previous approach using linearized tree, a degenerated version of tree composition.

Future work can be summarized in the following directions.

- **Better content modeling:** In our proposed syntax transfer task, we show that our model can perform excellent syntax transfer without parallel training data. However, the case analysis indicates that there still exist issues in current systems. We find that the critical challenge for the syntax transfer system is preserving content information as precisely as possible. The current systems, including our TA-VAE model, tend to have semantics drift during the generation by adding or neglecting words. This leads to a future research topic about how to perform precise content modeling.
- **Reliable evaluation datasets and metrics:** Though our TA-VAE model can train without access to parallel corpus, the evaluation set with golden transfer pairs is still necessary to measure the performance across different models. In this work, we adopt the word overlap and tree edit distance for the measurement. It will be better to have a standard evaluation set instead of only using those indirect metrics.
- **Unsupervised syntax modeling:** TA-VAE model still requires access to constituency tree labels, either from human ground truth or machine-generated. It will be better if we can do unsupervised tree composition.

References

[1] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 214–223, 2017.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[3] Hareesh Bahuleyan, Lili Mou, Hao Zhou, and Olga Vechtomova. Stochastic wasserstein autoencoder for probabilistic sentence generation. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4068–4076, 2019.

[4] Yu Bao, Hao Zhou, Shujian Huang, Lei Li, Lili Mou, Olga Vechtomova, Xinyu Dai, and Jiajun Chen. Generating sentences from disentangled syntactic and semantic spaces. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 6008–6019, 2019.

[5] Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 1171–1179, 2015.

[6] Ondrej Bojar, Christian Buck, Christian Federmann, Barry Haddow, Philipp Koehn, Johannes Leveling, Christof Monz, Pavel Pecina, Matt Post, Herve Saint-Amand, Radu Soricut, Lucia Specia, and Ale s Tamchyna. Findings of the 2014 workshop on statistical machine translation. In *Proceedings of the Ninth Workshop on Statistical Machine Translation*, pages 12–58, Baltimore, Maryland, USA, June 2014. Association for Computational Linguistics.

[7] Samuel Bowman, Luke Vilnis, Oriol Vinyals, Andrew Dai, Rafal Jozefowicz, and Samy Bengio. Generating sentences from a continuous space. In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, pages 10–21, 2016.

[8] Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 632–642, Lisbon, Portugal, September 2015. Association for Computational Linguistics.

[9] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. In *International Conference on Learning Representations*, 2018.

[10] Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. One billion word benchmark for measuring progress in statistical language modeling. In *Fifteenth Annual Conference of the International Speech Communication Association*, 2014.

[11] Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. Reading wikipedia to answer open-domain questions. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1870–1879, 2017.

[12] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, 2014.

[13] Jihun Choi, Kang Min Yoo, and Sang-goo Lee. Learning to compose task-specific tree structures. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[14] Christopher Clark and Matt Gardner. Simple and effective multi-paragraph reading comprehension. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 845–855, 2018.

[15] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[16] Adji B Dieng, Yoon Kim, Alexander M Rush, and David M Blei. Avoiding latent variable collapse with generative skip models. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 2397–2405, 2019.

[17] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[18] Kenneth Heafield. KenLM: Faster and smaller language model queries. In *Proceedings of the Sixth Workshop on Statistical Machine Translation*, pages 187–197, Edinburgh, Scotland, July 2011. Association for Computational Linguistics.

[19] Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. In *Advances in neural information processing systems*, pages 1693–1701, 2015.

[20] Zhiting Hu, Zichao Yang, Xiaodan Liang, Ruslan Salakhutdinov, and Eric P Xing. Toward controlled generation of text. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1587–1596, 2017.

[21] Vineet John, Lili Mou, Hareesh Bahuleyan, and Olga Vechtomova. Disentangled representation learning for non-parallel text style transfer. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 424–434, 2019.

[22] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[23] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[24] Thomas Laurent and James von Brecht. A recurrent neural network without chaos. *arXiv preprint arXiv:1612.06212*, 2016.

[25] Ruizhe Li, Xiao Li, Chenghua Lin, Matthew Collinson, and Rui Mao. A stable variational autoencoder for text modelling. In *Proceedings of the 12th International Conference on Natural Language Generation*, pages 594–599, 2019.

[26] Peter J Liu, Mohammad Saleh, Etienne Pot, Ben Goodrich, Ryan Sepassi, Lukasz Kaiser, and Noam Shazeer. Generating wikipedia by summarizing long sequences. In *International Conference on Learning Representations*, 2018.

[27] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[28] Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, pages 55–60, 2014.

[29] Mitch Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, 19(2):313–330, 1993.

[30] Tomáš Mikolov, Stefan Kombrink, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. Extensions of recurrent neural network language model. In *2011 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 5528–5531. IEEE, 2011.

[31] Tomáš Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 conference of the north american chapter of the association for computational linguistics: Human language technologies*, pages 746–751, 2013.

[32] Andriy Mnih and Karol Gregor. Neural variational inference and learning in belief networks. In *International Conference on Machine Learning*, pages 1791–1799, 2014.

[33] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318, 2002.

[34] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, 2016.

[35] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of the 31st International Conference on International Conference on Machine Learning-Volume 32*, pages II–1278, 2014.

[36] Alexander M Rush, Sumit Chopra, and Jason Weston. A neural attention model for abstractive sentence summarization. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 379–389, 2015.

[37] Tianxiao Shen, Tao Lei, Regina Barzilay, and Tommi Jaakkola. Style transfer from non-parallel text by cross-alignment. In *Advances in neural information processing systems*, pages 6830–6841, 2017.

[38] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems-Volume 2*, pages 3104–3112. MIT Press, 2014.

[39] Kai Sheng Tai, Richard Socher, and Christopher D Manning. Improved semantic representations from tree-structured long short-term memory networks. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1556–1566, 2015.

[40] Aaron Van Den Oord, Oriol Vinyals, et al. Neural discrete representation learning. In *Advances in Neural Information Processing Systems*, pages 6306–6315, 2017.

[41] Jiacheng Xu and Greg Durrett. Spherical latent spaces for stable variational autoencoders. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4503–4513, 2018.

[42] Kaizhong Zhang and Dennis Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM journal on computing*, 18(6):1245–1262, 1989.

[43] Junbo Zhao, Yoon Kim, Kelly Zhang, Alexander Rush, and Yann LeCun. Adversarially regularized autoencoders. In *International Conference on Machine Learning*, pages 5902–5911, 2018.

[44] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232, 2017.